Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services

Anthony Hock-Koon

UNIVERSITE DE NANTES
FACULTE DES SCIENCES ET DES TECHNIQUES

ÉCOLE DOCTORALE STIM

« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATHÉMATIQUES »

Année 2011

N° attribué par la bibliothèque

Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services

THÉSE DE DOCTORAT

Discipline : Informatique

Spécialité : Génie logiciel

Présentée
et soutenue publiquement par

Anthony HOCK-KOON

Le 20 septembre 2011 à l'UFR Sciences et Techniques, Université de Nantes,
devant le jury ci-dessous

Président	:	_____	_____
Rapporteurs	:	Djamal BENSLIMANE, Professeur	Université Claude Bernard Lyon
		Lionel SEINTURIER, Professeur	Université de Lille
Examinateurs	:	Pascal MOLLI, Professeur	Université de Nantes
		Jean-Claude ROYER, Professeur	École des Mines de Nantes
		Arnaud GERARD,	Responsable AG conseil

Directeur de thèse : Mourad OUSSALAH

Laboratoire : **L**ABORATOIRE D'**I**NFORMATIQUE DE **N**ANTES **A**TLANTIQUE.
UMR CNRS 6241. 2, rue de la Houssinière, BP 92 208 – 44 322 Nantes, CEDEX 3.

N° ED

CONTRIBUTION À LA COMPRÉHENSION ET À LA MODÉLISATION DE LA COMPOSITION ET DU COUPLAGE FAIBLE DE SERVICES DANS LES ARCHITECTURES ORIENTÉES SERVICES

Contribution to the understanding and modeling of the composition and loose coupling of services in Service-Oriented Architectures

Anthony Hock-koon

favet neptunus eunti

Université de Nantes

Anthony Hock-koon

Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services

195 pages

Table des matières

Table des matières **i**

Table des figures **vii**

Liste des tableaux **ix**

Introduction **1**

1 Ingénierie logicielle orientée services et Composition de services **9**
1.1 Introduction . . . 9
1.2 Origines conceptuelles . . . 10
1.2.1 Idée directrice . . . 10
1.2.2 Cibles d'applications . . . 11
1.2.3 La notion de Service . . . 11
1.2.3.1 Service et description de service . . . 12
1.2.3.2 Architecture orientée services : AOS . . . 12
1.3 Mécaniques opératoires . . . 14
1.3.1 Publication de services . . . 14
1.3.2 Composition de services . . . 15
1.3.2.1 Découverte de services . . . 15
1.3.2.2 Sélection et négociation de services . . . 16
1.3.2.3 Composition de services . . . 17
1.4 Composition dynamique de services . . . 17
1.4.1 Collaboration entre services . . . 18
1.4.1.1 Orchestration et chorégraphie . . . 18
1.4.1.2 Problématique de la collaboration : l'exemple des services Web . . . 19
1.4.1.3 Références illustratives . . . 21
1.4.2 Gestion des hétérogénéités . . . 23
1.4.2.1 Aspects technologiques, conversationnels, et de données . . . 23
1.4.2.2 Références illustratives . . . 24
1.4.3 Adaptation contextuelle d'une composition de services . . . 25

1.4.3.1 Fonctionnement global . . . 26
1.4.3.2 Références illustratives . . . 27
1.4.4 Vers notre approche de méta-modélisation d'un service composite . 28
1.4.4.1 Offrir une meilleure compréhension de la composition . . . 29
1.4.4.2 Problématiques additionnelles à la notion de service composite . . . 30
1.5 Conclusion . . . 31

2 Composition de services et méta-modèle de service composite 33
2.1 Introduction . . . 33
2.2 Éléments architecturaux . . . 35
2.2.1 Vue globale . . . 35
2.2.2 GSC : Gestionnaire de service composite . . . 37
2.2.3 Gestionnaire de collaboration . . . 38
2.2.3.1 Type et instance de schéma de collaboration . . . 38
2.2.3.2 Service abstrait et service concret . . . 38
2.2.3.3 Vers l'héritage de type de schéma de collaboration . . . 39
2.2.3.4 Gestionnaire d'observation des exécutions . . . 41
2.2.4 Gestionnaire d'hétérogénéités . . . 42
2.2.5 Gestionnaire de découverte et sélection . . . 44
2.2.6 Gestionnaire multitenant . . . 45
2.3 Mécanismes et dépendances . . . 45
2.3.1 Mécanique opératoire sans défaillance . . . 46
2.3.1.1 Exécution simple . . . 46
2.3.1.2 Exécution multitenant . . . 47
2.3.2 Dépendances entre gestionnaires . . . 50
2.3.2.1 Du gestionnaire de découverte et sélection vers le gestionnaire de collaboration . . . 50
2.3.2.2 Du gestionnaire de découverte et sélection vers le gestionnaire d'hétérogénéités . . . 51
2.3.3 Le processus d'auto-composition . . . 52
2.3.4 Vers la réduction du couplage . . . 53
2.3.4.1 Bénéfices du couplage faible . . . 53
2.3.4.2 Gestion du couplage faible dans le méta-modèle . . . 54
2.4 Exemple d'instanciation . . . 54
2.4.1 Projection du service composite . . . 55
2.4.2 Exécution sans défaillance . . . 56
2.4.3 Auto-composition . . . 57
2.5 Conclusion . . . 59

3 Vers une nouvelle définition du couplage faible 61
3.1 Introduction . . . 61
3.2 Motivations et objectifs . . . 62

3.2.1 Origine des confusions . . . 62
3.2.2 Imprécisions des métriques existantes . . . 63
3.2.3 Expliciter le couplage . . . 64
3.3 La notion de couplage faible . . . 64
3.3.1 Localisation des dépendances . . . 64
3.3.1.1 Vers trois couplages distincts . . . 64
3.3.1.2 Exemple . . . 66
3.3.2 Le couplage sémantique . . . 67
3.3.3 Le couplage syntaxique . . . 69
3.3.4 Le couplage physique . . . 71
3.4 Formule d'évaluation globale du couplage d'un composite . . . 73
3.4.1 Criticité : A . . . 73
3.4.2 Probabilité d'occurrence d'une défaillance : B . . . 75
3.4.3 Probabilité de non détection d'une défaillance : C . . . 78
3.5 Comparer des approches de composition . . . 79
3.5.1 Critères de comparaison . . . 80
3.5.1.1 Abstraction du couplage sémantique . . . 80
3.5.1.2 Abstraction du couplage syntaxique . . . 80
3.5.1.3 Abstraction du couplage physique . . . 83
3.5.1.4 Abstraction de la formule d'évaluation globale du couplage 84
3.5.2 Cadre de comparaison . . . 85
3.6 Conclusion . . . 87

4 CBSE vs SOSE : vers un cadre conceptuel de comparaison 89
4.1 Introduction . . . 89
4.2 Analyse de l'existant . . . 91
4.2.1 Origines des confusions . . . 91
4.2.1.1 Confusions conceptuelles . . . 91
4.2.1.2 Confusions techniques . . . 92
4.2.2 Limitations des comparaisons existantes . . . 93
4.2.2.1 Comparaisons intra-paradigmes . . . 94
4.2.2.2 Comparaisons inter-paradigmes . . . 94
4.2.3 Objectifs de ce cadre conceptuel de comparaison . . . 96
4.3 Différences conceptuelles . . . 97
4.3.1 Différence d'usage et de responsabilité propriétaire . . . 97
4.3.1.1 Responsabilité sur un composant . . . 98
4.3.1.2 Responsabilité sur un service . . . 99
4.3.1.3 Nature multitenant du service . . . 100
4.3.2 Différence de rapport au couplage . . . 101
4.3.2.1 Gestion des hétérogénéités . . . 101
4.3.2.2 Rapport à l'automatisation des mécanismes . . . 101
4.3.3 Différence de granularité . . . 102
4.3.4 Principales préoccupations de OO, CBSE et SOSE . . . 103

4.4 Aspects quantitatifs . . . 105
4.4.1 Produits et processus . . . 105
4.4.2 Comparaison entre objet, composant et service . . . 106
4.4.2.1 Produit . . . 106
4.4.2.2 Processus . . . 107
4.5 Aspects qualitatifs . . . 110
4.5.1 Propriétés qualitatives de comparaison des paradigmes de développement . . . 110
4.5.2 Comparaison Objet, Composant et Service . . . 111
4.5.3 Perspectives d'analyse qualitative . . . 113
4.5.3.1 Définition et modélisation d'une perspective qualitative . . 113
4.5.3.2 Exemple de perspectives qualitatives : réutilisabilité, composabilité, dynamicité . . . 114
4.6 Retour aux contributions à SOSE à travers le cadre de comparaison . . . 116
4.6.1 Amélioration de l'évaluation qualitative : l'exemple du couplage faible 116
4.6.1.1 Redéfinition du couplage faible . . . 117
4.6.1.2 Perspectives d'amélioration du cadre conceptuel . . . 117
4.6.2 Rapport du méta-modèle de service composite face au SOSE . . . 118
4.6.2.1 Aspects quantitatifs . . . 118
4.6.2.2 Aspects qualitatifs . . . 121
4.6.2.3 Vers une méta-modélisation par services . . . 122
4.6.2.4 Amélioration du méta-modèle de service composite . . . 124
4.7 Conclusion . . . 124

5 Réalisation et Expérimentation du méta-modèle 127
5.1 Introduction . . . 127
5.2 Scénarii de domotique . . . 128
5.2.1 Écoute de musique et vidéo-conférence . . . 128
5.2.2 Contraintes d'exécutions . . . 129
5.2.2.1 Contraintes simples . . . 129
5.2.2.2 Contraintes complexes . . . 129
5.3 Instanciation du méta-modèle . . . 130
5.3.1 Service composite : Commande vocale . . . 130
5.3.1.1 Fonctionnement métier . . . 130
5.3.1.2 Modélisation du service composite Commande vocale . . . 131
5.3.2 Service composite : Écoute de musique . . . 135
5.3.2.1 Fonctionnement métier . . . 135
5.3.2.2 Modélisation du service composite Écoute de musique . . 136
5.3.2.3 Service composite : Exécution_écoute . . . 137
5.3.3 Service composite : Vidéo-conférence . . . 138
5.4 Réalisation en SCA et Java . . . 138
5.4.1 Présentation de SCA . . . 138
5.4.2 Restrictions par rapport aux modèles . . . 140

5.4.3 Implémentation 140
5.5 Conclusion 143

Conclusion et perspectives **144**

Liste des publications de nos travaux **151**

Bibliographie **153**

Annexe **165**

A Retour au cadre conceptuel : évaluation de Fractal, SCA et OSGi **1**
A.1 Introduction 1
A.2 Le modèle Fractal 2
A.3 Le modèle SCA 3
A.3.1 SCA Tuscany 3
A.3.2 SCA FraSCAti 5
A.4 Le modèle OSGi 6
A.5 Conclusion 9

B Découverte de services M-N : vers une première proposition **11**
B.1 Introduction 12
B.2 Vue globale des approches de découverte de services 12
B.2.1 Classification de l'existant 13
B.2.2 Vers une correspondance M-N 13
B.2.2.1 Principes de base 14
B.2.2.2 Bénéfices attendus 14
B.2.2.3 Imbrications dans les travaux de thèse 15
B.3 Présentation de l'approche M-N 15
B.3.1 La notion de famille de services 16
B.3.2 Graphes de familles 17
B.3.3 Identification de compositions concrètes 18
B.3.4 Vers la définition automatisée de nouvelles familles 19
B.4 Conclusion 19

Table des figures

0.1 Organisation du travail de recherche et des contributions 5
0.2 Représentation schématique de la structure globale de la thèse 7

1.1 Organisation d'une architecture orientée services 13
1.2 Orchestration vs Chorégraphie de services 19
1.3 Technologies de collaboration de services Web 21

2.1 Méta-modèle de service composite : vue globale 36
2.2 GSC : gestionnaire de service composite 37
2.3 Gestionnaire de collaboration : graphes 41
2.4 Gestionnaire d'observation : graphe 42
2.5 Gestionnaire d'hétérogénéités : graphe 43
2.6 Gestionnaire de découverte et sélection : graphe 44
2.7 GSC et services gestionnaires 46
2.8 Exécution simple 47
2.9 Exécution multitenant 49
2.10 Scénario domotique : graphes des services gestionnaires 56
2.11 Héritage et instanciation de type de schéma de collaboration 58
2.12 Exemple d'instanciation : processus d'auto-composition 59

3.1 Cibles des couplages sémantique, syntaxique et physique 65
3.2 Modélisation du service composite Voiture 67
3.3 Couplage sémantique 69
3.4 Couplage syntaxique 70
3.5 Couplage physique 72
3.6 Couplage physique d'une orchestration 72
3.7 Répartition des résultats des formules de criticité 74
3.8 Natures des correspondances 1-N 82

4.1 Évolution des préoccupations globales entre objet, composant et service 104
4.2 Niveaux d'abstraction et de description : répartition des produits et des processus 106
4.3 Comparaison des propriétés entre objet, composant et service 111
4.4 Modélisation d'une perspective qualitative 114

4.5 Expression des perspectives de réutilisabilité, composabilité et dynamicité . . . 115
4.6 Répartition des produits et processus du méta-modèle de service composite . . 120
4.7 Synthèse des apports qualitatifs du méta-modèle de service composite 122
4.8 Hiérarchie de méta-modélisation composant et service 123
4.9 Fonctionnement global du cadre conceptuel de comparaison 125

5.1 Diagramme de séquence : fonctionnement du service composite *Commande vocale* 131
5.2 Modélisation du service composite *Commande vocale* 132
5.3 Service composite *Commande vocale* : graphes des services gestionnaires 133
5.4 Diagramme de séquence : fonctionnement du service composite *Commande vocale* avec services gestionnaires 134
5.5 Diagramme de séquence : fonctionnement du service composite *Écoute de musique* 135
5.6 Service composite *Écoute de musique* : graphes des services gestionnaires . . . 137
5.7 Service composite SCA 139
5.8 Capture d'écran : implémentation SCA du service composite *Écoute de musique* 141
5.9 Capture d'écran : séquence de déplacement 142
5.10 Capture d'écran : séquence de défaillance 142

A.1 Approche Fractal dans le CBSE 4
A.2 Approche SCA : entre CBSE et SOSE 5
A.3 Comparaison SCA Tuscany et SCA FraSCAti 7
A.4 Approche OSGi dans le SOSE 8

B.1 Classification des algorithmes de découverte de services 13
B.2 Vers une correspondance M-N 14
B.3 Réorganisation de schéma de collaboration 15
B.4 Exemple de graphe de familles de services 17
B.5 Graphe de familles de services et réorganisation de schéma 18

Liste des tableaux

1.1 Écoles européennes, américaines et asiatiques, et françaises 29

3.1 Exemple de calcul de la criticité 76
3.2 Cadre de comparaison d'approches de composition de services 86

4.1 CBSE vs SOSE : répartition des responsabilités 100
4.2 Produit-Processus : comparaison entre objet, composant et service 108
4.3 Produits et processus du méta-modèle de service composite 119
4.4 Expression de la composabilité et de la dynamicité 121

A.1 Fractal : produits et processus 3
A.2 SCA : produits et processus 4
A.3 SCA FraSCAti : produits et processus 6
A.4 OSGi : produits et processus 8

Introduction

Cadre de la thèse

Cette thèse s'inscrit dans le cadre de l'ingénierie logicielle orientée services [SD05] (SOSE - Service-Oriented Software Engineering) qui est un paradigme récent de développement logiciel. Elle aborde plus précisément, la notion d'architectures orientées services [OAS08, PH07] (AOS) et le processus associé de composition de services [YP04, NGM+08] qui sont des composantes du SOSE.

Un paradigme de développement logiciel traite de la manière dont une solution informatique à un problème doit être formulée suivant une liste bien définie de concepts et de mécanismes. Il détermine la vue dans la façon d'appréhender ce problème et fournit les moyens d'exploiter cette vue, de suivre ses principes et de la réaliser concrètement. Ainsi, un paradigme de développement logiciel correspond à un style particulier d'élaboration de solutions informatiques, en termes d'analyse, de modélisation et de réalisation. Par nature, un paradigme est indépendant de problèmes métiers spécifiques, mais il peut favoriser certains types d'applications afin de supporter des qualités particulières. Cependant, ces qualités particulières sont en général associées à des contre-coups particuliers.

Le SOSE est un style distinctif d'élaboration de solutions informatiques qui cible le développement d'applications devant supporter des environnements hautement volatiles et hétérogènes. Un architecte logiciel, qui suit les principes SOSE, construit son application en réutilisant des services existants, préalablement déployés sur le réseau, et en combinant leurs fonctionnalités pour réaliser les tâches voulues. Ce principe d'assemblage pose deux problématiques principales : l'identification des services qui correspondent aux besoins de l'architecte, et la définition de la collaboration entre ces services.

Ainsi, le SOSE introduit les notions de services et d'architectures orientées services qui posent un ensemble de principes guides pour construire ces applications par assemblages de services.

Le service correspond à un bloc logiciel fonctionnel qui représente l'élément architectural de base d'une architecture orientée services. Son rôle est d'encapsuler de façon homogène tout type de ressources quelque soit leurs hétérogénéités de fonctions ou d'implémentations. Ensuite, le service est spécifié de façon à supporter la volatilité des environnements ciblés par le SOSE. Il a vocation à être dynamiquement incorporé ou retiré des applications afin d'automatiser leur développement et leurs modifications pour

répondre rapidement aux évolutions des besoins.

L'AOS fait intervenir les concepts de fournisseurs et de consommateurs de services. Le fournisseur correspond à l'acteur qui cherche à promouvoir l'utilisation des ressources encapsulées dans son service. Les consommateurs sont les acteurs qui exploitent ces ressources en fonction de leurs besoins. Lors du développement de son service, le fournisseur n'a pas connaissance des consommateurs tiers qui l'utiliseront effectivement, ni de ce qu'ils cherchent à obtenir via son exploitation. Cette organisation est la base de l'automatisation des constructions de nouvelles applications. Elle induit la problématique d'établissement dynamique de la relation d'utilisation entre fournisseur et consommateurs. Pour établir cette relation, le SOSE repose sur deux processus : la *publication de services* et la *composition de services*.

La *publication de services* est sous la responsabilité du fournisseur de services qui doit s'assurer de la visibilité de son service dans un marché d'offres concurrentielles. Pour ce faire, il doit dans un premier temps décrire rigoureusement les fonctionnalités fournies par son service et la manière de communiquer avec lui pour les exploiter. Puis, il doit mettre ces informations à la portée des consommateurs de façon à ce qu'ils puissent les trouver et ainsi utiliser le service associé si ce dernier correspond à leurs besoins.

De son côté, le consommateur est donc responsable de cette réutilisation qui repose sur le processus de *composition de services*. Ce processus implique différentes phases où le consommateur doit i) comprendre les offres existantes de services afin d'identifier ceux qui correspondent le plus à ses besoins, puis, ii) il doit être capable de définir et de gérer leurs collaborations pour réaliser l'application finale.

Dans cette thèse, nous nous intéressons particulièrement à ce processus de composition de services et à son automatisation.

Problématique abordée

La maîtrise de la composition de services passe obligatoirement par la compréhension des principes SOSE, de son approche particulière du développement logiciel et de ses enjeux en terme de qualités attendues. Mieux cerner ces objectifs nous a permis de mettre en évidence un certain nombre de limitations dans les propositions actuelles de composition de services.

De fait, la composition de services est un processus complexe qui fait intervenir un ensemble d'étapes intermédiaires. Ces étapes se répartissent suivant les préoccupations classiques liées à la réutilisation d'entités logicielles existantes : i) l'identification des services qui seront composés, puis ii) la réalisation effective de cette composition. Ainsi, la communauté service propose pour chacune de ces étapes une multitude d'approches différentes qui ont des perspectives et des préoccupations particulières. L'objectif final est d'atteindre une capacité d'adaptation automatisée et autonome d'une composition de services face aux changements de contextes afin de supporter le développement d'applications liées à des environnements hautement volatiles.

Notre travail de recherche s'inscrit dans cette même problématique. Après avoir

étudié les méthodes existantes, nous proposons une approche différente de la composition dynamique de services.

Ainsi, nous essayons d'améliorer le processus de composition dynamique sur un aspect particulier : la réduction du couplage entre les services réutilisés. Réduire ce couplage c'est réduire les dépendances dans la composition de services et donc faciliter les modifications et améliorer la robustesse face aux défaillances. Cependant, une définition claire du couplage faible est toujours absente et il en résulte des métriques [PRFT07,GS08,EKM07,MZZW09] qui ne permettent pas d'appréhender et de hiérarchiser les travaux existant sur la réduction du couplage.

Ainsi, un autre aspect de notre travail de thèse est d'offrir une meilleure compréhension de la notion de couplage faible. Nous proposons de nouvelles méthodes de mesure du couplage afin d'établir les apports des approches existantes, et ainsi les comparer avec notre travail.

En parallèle, notre compréhension croissante des concepts du SOSE a rapidement mis en évidence une similarité forte entre composants et services. En effet, l'ingénierie logicielle à base de composants [HC01, Szy02] (CBSE - Component based software engineering) partage avec le SOSE de nombreuses problématiques. Cette jonction de problématiques vient de leur noyau conceptuel commun où une application est construite par composition d'entités logicielles existantes, composants ou services. CBSE et SOSE font donc face à des challenges similaires.

Malgré son importance, la question de la différence claire entre CBSE et SOSE est encore sans réponse satisfaisante. Après avoir présenté notre travail dans le SOSE sur la composition de services et le couplage faible, nous étendons notre état de l'art afin de nous confronter à cette problématique. Cette thèse essaie d'apporter une meilleure compréhension du SOSE par sa mise en perspective face au CBSE. Cette problématique de comparaison entre paradigmes nécessite la construction d'un référentiel haut niveau commun, capable de manipuler composant et service sans favoriser l'un ou l'autre.

Contributions

Les contributions de cette thèse s'organisent suivant les résultats de trois axes de recherche :

- la définition d'un méta-modèle de service composite comme proposition à la composition dynamique de services ;
- la redéfinition et une nouvelle proposition de la notion de couplage faible, pour permettre la construction de nouvelles métriques ; et
- le cadre conceptuel de comparaison entre CBSE et SOSE, pour disposer d'un référentiel de comparaison des deux paradigmes et leur apporter une meilleure compréhension.

La **première contribution** est la spécification d'un méta-modèle de service composite. L'approche choisie dans ce méta-modèle est de réifier au niveau architectural un certain nombre de caractéristiques liées au processus de composition issues des travaux

existants [BGPT09,YS07,SMF+09,CNP09,FFL10]. Ces caractéristiques sont sélectionnées en fonction des deux objectifs de départ du méta-modèle :

- le support des adaptations dynamiques contextuelles de la composition de services encapsulée dans le composite ; et
- la minimisation du couplage entre les services de cette composition.

Le méta-modèle de service composite permet de mieux comprendre le processus de composition à travers l'abstraction de caractéristiques précises et l'expression de leurs inter-dépendances.

La validation de notre méta-modèle passe en partie par la mesure de ces apports sur la réduction du couplage. Cependant, l'étude bibliographique que nous avons menée a rapidement mis en évidence les limites, d'une part dans la définition de la notion de couplage, et d'autre part, dans les métriques associées qui ne prennent pas en compte les mécanismes haut-niveau, obligatoires et spécifiques au SOSE.

La **deuxième contribution** correspond donc à la proposition d'une nouvelle définition de la notion de couplage faible qui soit plus adaptée à la réalité du fonctionnement du SOSE. Cette définition repose sur la proposition de trois nouveaux couplages nommés : *couplage sémantique*, *couplage syntaxique*, et *couplage physique*. Ce découpage est à la base de nouvelles métriques qui sont combinées dans une formule inspirée du domaine de l'analyse préliminaire de risques [Mor02] pour évaluer le couplage d'une composition cible. De plus, cette redéfinition du couplage permet la construction d'un cadre de comparaison d'approches de composition qui les classifie en fonction de leur potentiel de réduction des dépendances. Ce cadre est aussi utilisé pour évaluer notre méta-modèle de service composite.

La **troisième contribution** fait un retour sur l'état de l'art du SOSE et le met en perspective par rapport au CBSE. Concrètement, nous spécifions un cadre conceptuel de comparaison qui est capable de manipuler composant et service sans prendre de points de vue qui avantageraient l'un ou l'autre des paradigmes. Afin d'atteindre cet objectif, les différentes catégories qui composent notre cadre sont définies indépendamment de tout paradigme de développement logiciel. Cette approche nous permet d'inclure dans le même cadre la comparaison avec l'orienté objet [Oa99] (OO).

Le cadre conceptuel de comparaison se décompose en deux aspects :

- l'aspect quantitatif : qui identifie et classifie les supports mis en place par les différents paradigmes pour atteindre leurs objectifs de qualités ; et
- l'aspect qualitatif : qui spécifie ces objectifs de qualités à travers un ensemble de propriétés permettant de les analyser suivant des perspectives personnalisées, en fonction de l'expertise de l'utilisateur.

Ainsi, nous essayons d'apporter une vision plus globale qui appréhende l'évolution des préoccupations du génie logiciel à travers les paradigmes objet, composant et service.

La figure 0.1 résume les contributions suivant les trois axes de recherche et leurs articulations les uns par rapport aux autres.

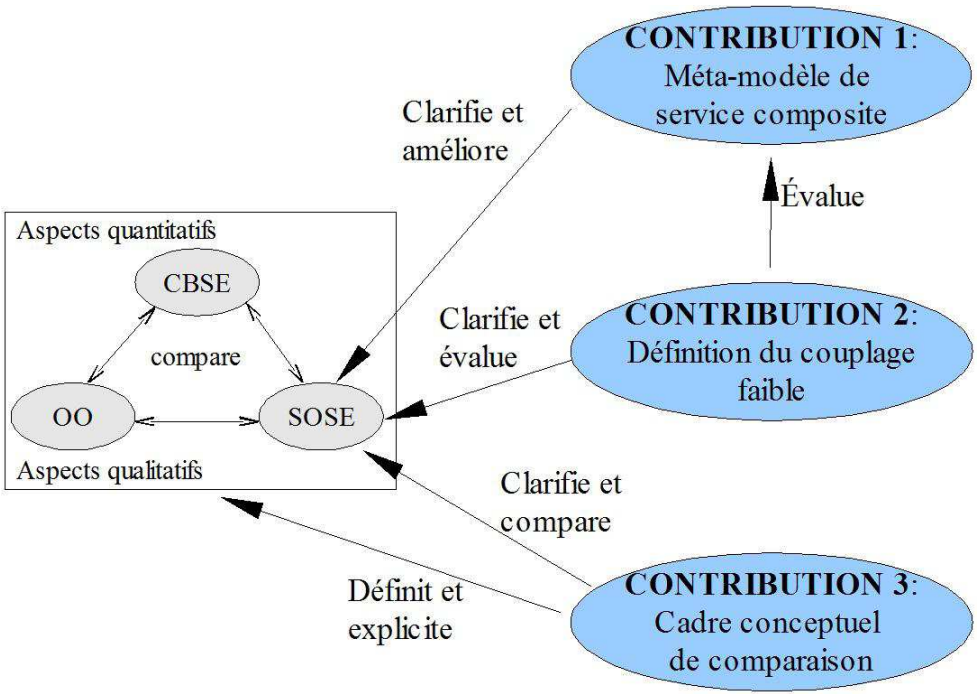

Fig. 0.1 – Organisation du travail de recherche et des contributions

Organisation du document

Cette thèse est composée de cinq chapitres :

Le chapitre 1 pose le socle bibliographique de notre travail. Il propose un état du domaine du SOSE à travers le prisme des AOS et du mécanisme de composition de services. Dans un premier temps, nous revenons sur les origines du SOSE et les motivations qui ont poussé son développement. Nous définissons clairement les concepts de service et d'architecture orientée services (AOS), puis nous proposons une vue globale des principaux mécanismes associés. Dans un second temps, nous mettons l'accent sur un de ces mécanismes en particulier : le processus de composition de services. Nous présentons un panel choisi des approches existantes afin d'illustrer les optiques d'étude majeures proposées dans la littérature. Nous identifions en particulier la recherche autour de la composition dynamique de services à travers certains travaux des communautés américaines et asiatiques, européennes, et françaises.

Nous concluons ce chapitre par l'identification des problématiques que notre travail sur la définition d'un méta-modèle de service composite essaie de résoudre.

Le chapitre 2 décrit notre méta-modèle de service composite. L'approche choisie est de réifier au niveau architectural certaines caractéristiques associées au processus de composition afin de permettre la réduction du couplage. Cette réification identifie différents éléments architecturaux qui construisent notre vision du service composite. Puis, nous caractérisons les dépendances entre ces éléments et spécifions les mécanismes

qui régissent leurs interactions. Ce travail aboutit à la proposition du processus d'auto-composition dans notre service composite qui assure ses adaptations dynamiques aux changements de contextes.

Le chapitre 3 met en évidence les limitations des définitions existantes du couplage faible dans une composition de services. Puis, il présente notre nouvelle définition à travers les couplages sémantique, syntaxique et physique et leurs métriques associées. Enfin, il détaille comment ces métriques sont combinées pour définir une formule d'évaluation du couplage d'une composition ciblée, et comment elles permettent la construction d'un cadre de comparaison d'approches de composition.

Le chapitre 4 change de perspective sur l'état de l'art en traitant la question récurrente de la définition claire d'un service par rapport à un composant et vice-versa. Dans un premier temps, nous mettons en évidence l'absence d'études approfondies sur cette problématique. Nous discutons ensuite des différences au niveau paradigme entre CBSE et SOSE. Puis, nous présentons le cadre conceptuel de comparaison découpé en aspects quantitatifs et qualitatifs. Cette comparaison prend en compte l'OO afin d'offrir un point de vue plus global sur l'évolution des préoccupations entre objet, composant et service. Enfin, nous réutilisons ces aspects quantitatifs et qualitatifs pour déterminer les contributions au SOSE apportées par le méta-modèle de service composite et par le travail sur le couplage faible.

Le chapitre 5 revient sur le méta-modèle de service composite et présente une implémentation de ses principes en utilisant les technologies Java et SCA [OAS09]. L'objectif de ce chapitre est de démontrer la faisabilité technique des propositions faites dans les chapitres précédents.

En guise de conclusion, nous faisons le bilan précis de ce travail en soulignant nos apports principaux et en mettant en valeur leurs contributions dans le domaine du SOSE. De plus, nous faisons un retour critique sur la recherche effectuée durant cette thèse et nous suggérons un ensemble de pistes d'extensions possibles.

La structure globale de la thèse est résumée dans la figure 0.2.

Le document se termine par deux annexes. L'annexe A réutilise le cadre conceptuel de comparaison et l'applique pour évaluer les technologies Fractal [BCL+06], SCA [OAS09] et OSGi [OSG11]. Enfin, l'annexe B présente une nouvelle proposition d'algorithme de découverte de services.

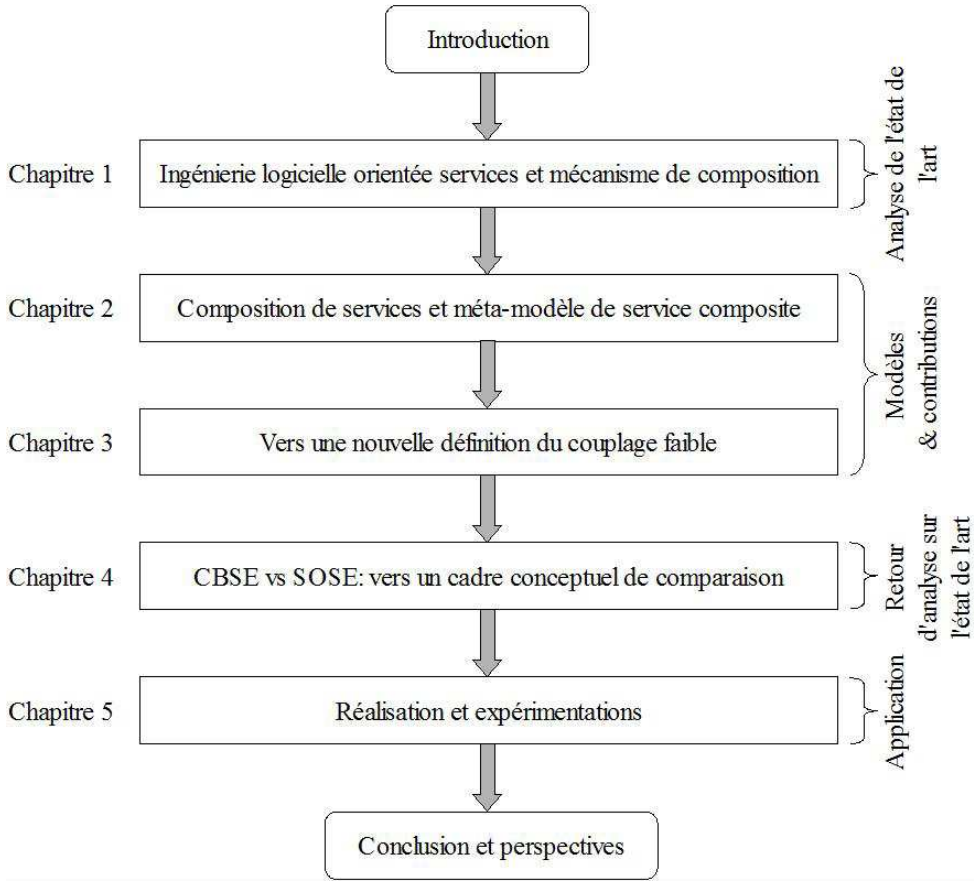

Fig. 0.2 – Représentation schématique de la structure globale de la thèse

Chapitre 1

Ingénierie logicielle orientée services et Composition de services

1.1 Introduction

L'ingénierie logicielle orientée services (Service-Oriented Software Engineering - SOSE [SD05, The08]) est un paradigme de développement logiciel relativement récent, datant du début des années deux mille, et bien établi dans le domaine. Le SOSE est directement inspiré des modes d'organisations commerciales réelles entre multinationales, et se base sur leur notion classique de service offert. Il reprend les principes de la mondialisation, vers la sous-traitance et la réduction maximale des coûts de production où une entreprise se focalise sur des fonctionnalités innovantes qui font sa plus value par rapport aux autres, et favorise l'externalisation des fonctionnalités à faible valeur ajoutée. Un exemple très simple de cette réalité peut être donné par des constructeurs de cartes graphiques tels que Nvidia[1] ou ATI[2] dont la valeur ajoutée est dans le développement d'architectures performantes, et non dans la production et l'assemblage de ces cartes qui sont sous-traitées.

L'origine de l'orienté service vient de demandes grandissantes liées à des systèmes devant supporter des environnements de plus en plus volatiles et hétérogènes tels que l'Internet et les services Web [ACKM03], les environnements d'intelligence ambiante (Iam [Wei91]) ou les applications business diffusées sur des réseaux d'entreprises telles que les

[1] http ://www.nvidia.com/
[2] http ://www.amd.com/

ERP [TIOP01, PH07]. Quelque soit le domaine d'application, la rapidité est l'élément clé du profit. La productivité d'un fournisseur et sa réactivité aux changements de besoins représentent des enjeux majeurs dont le SOSE tente d'apporter des solutions dans le développement logiciel.

Dans ce chapitre, nous présenterons les techniques et les méthodologies mises en place pour supporter les principes SOSE. La section 1.2 présente les origines conceptuelles du paradigme, et explicite les notions clés de service et d'architecture orientée services (AOS [OAS08, PH07]). La section 1.3 décrit les principaux mécanismes liés au service qui permettent le développement de nouvelles applications et leurs exécutions. Dans la section 1.4, nous nous focalisons sur le mécanisme de composition de services et sur les différents aspects qui le caractérisent. Chaque aspect est détaillé par une sélection de travaux proposés par les communautés "service" américaines et asiatiques, européennes, et françaises. Enfin, la section 1.5 conclut le chapitre.

1.2 Origines conceptuelles

1.2.1 Idée directrice

Le SOSE part d'une idée simple, inspirée par la mondialisation dans le commerce actuel, qui repose principalement sur la sous-traitance. Une société préfére se concentrer sur des activités à forte valeur ajoutée qui représentent son coeur de métier et découlent de son expertise singulière. C'est cette particularité par rapport aux autres entreprises concurrentes qui sera porteuse de bénéfices. Les activités plus basiques ou hors de sa maîtrise sont déléguées à d'autres sociétés.

La sous-traitance a de nombreux avantages :

- l'augmentation de la productivité : la société n'a pas à réaliser elle-même certaines activités qui sont sous la responsabilité des sous-traitants. Elle économise le temps et les efforts nécessaires à la mise en place de solutions pour ces activités.
- la baisse des coûts : gagner du temps permet une mise sur le marché du produit plus tôt et donc, de "grignoter" des parts de marché sur la concurrence. De plus, le choix de sous-traiter une activité se fait, en général, si son coût est moins important qu'une réalisation en interne.
- la réactivité accrue : l'évolution de la production en fonction de nouveaux besoins est plus rapide et plus simple. Si des sous-traitants ne remplissent plus les nouvelles conditions, la société a juste à en changer et en sélectionner d'autres. La dynamicité dans la chaîne de production est améliorée.

C'est la volonté d'obtenir les mêmes avantages dans l'ingénierie logicielle qui sera le postulat de départ du paradigme SOSE.

Ainsi un développeur qui définit son application suivant les concepts du SOSE cherchera à réutiliser des fonctionnalités offertes par des fournisseurs tiers, à travers des entités logicielles appelées **services**. Son application sous-traitera tout ou partie de son fonctionnement à ces services. Le SOSE fait intervenir les notions de *consommateurs* et

de *fournisseurs de services*, c'est-à-dire ceux qui consomment les services et ceux qui les mettent à disposition.

1.2.2 Cibles d'applications

Les concepts SOSE ciblent le développement d'applications dont l'environnement d'exécution est hautement volatile et hétérogène. Ainsi, ces applications doivent supporter des propriétés spécifiques et obligatoires pour gérer de tel environnement et être viables en termes d'utilisation et de qualité de service.

La volatilité d'un environnement correspond à la fréquence élevée des évolutions et des changements de contextes, que ces contextes soient liés aux acteurs, consommateurs ou fournisseurs, ou à l'environnement. Ces évolutions peuvent être voulues ou non voulues par rapport à un acteur particulier, et être prévues ou non prévues par ces même acteurs.

Les évolutions peuvent être regroupées suivant leurs cibles d'action [Pap08] :

- les *évolutions des besoins* : au niveau du consommateur qui change d'exigences sur les services réutilisés (modifications des contraintes de fonctionnalité, de qualité de services, etc), ou au niveau du fournisseur qui veut changer de cibles métiers en modifiant les aspects fonctionnels ou non fonctionnels de son service ; et
- les *évolutions dues à des défaillances* : quelque soit le type de ces défaillances qui peuvent être au niveau du réseau, des messages échangés, du service en lui même, etc.

L'hétérogénéité de l'environnement est directement liée à l'indépendance des fournisseurs entre eux, et par rapport aux consommateurs, où le développement de leurs services est isolé. Les hétérogénéités se retrouvent à différents niveaux [RHL09, OAS08, PH07] :

- *fonctionnelles et non fonctionnelles* : le fournisseur définit son service en termes de fonctionnalités offertes et de qualité de services non fonctionnelle (sécurité, performances, politiques, etc.) qui peuvent ne pas correspondre exactement aux besoins. De fait, il n'a pas de connaissances à priori précises sur les clients potentiels et sur la façon dont ils projettent de réutiliser son service ; et
- *réalisation* : les fournisseurs peuvent implémenter leurs services suivant différentes technologies, et donc être accessibles aux utilisations via des contraintes différentes sur les protocoles de communication, les aspects conversationnels ou types de données échangées, etc.

Le SOSE propose un ensemble de solutions de modélisation et de réalisation d'applications qui peuvent supporter cette volatilité des contextes et l'hétérogénéité des ressources qu'elles utilisent. Le socle conceptuel sur lequel repose toute la construction du SOSE est l'encapsulation de ces ressources hétérogénes sous la notion de *Service*.

1.2.3 La notion de Service

Le service est la brique de base du SOSE. Il représente une entité logicielle fonctionnelle déployée et invocable à distance. Un service est lié à un fournisseur de services et est utilisé

par des consommateurs de services.

1.2.3.1 Service et description de service

Un service assure un nombre défini de fonctionnalités. Il est vu comme une boite noire [OAS08, Erl05] pour ses consommateurs, et a un fonctionnement auto-contenu et indépendant. Il peut être "*stateless*" ou "*stateful*" :

- *stateless* : sans états internes où chaque exécution est monolithique et indépendante, par exemple un service de traduction français-anglais.
- *stateful* : avec états internes, c'est-à-dire que ses différentes exécutions dépendent des précédentes. Typiquement, un service de gestion de comptabilité d'entreprises.

Un service est obligatoirement *multitenant* [Jac05], c'est-à-dire capable de gérer de multiples consommateurs en parallèle. Cette propriété est plus complexe dans le cas d'un service stateful, car il doit manipuler de nombreux contextes clients en parallèle et garantir leur cohérence et leur isolation au fil des exécutions.

Un service est lié à une *description de service* qui renseigne sur son interface et sur la façon de communiquer avec lui.

La *description de service* décrit les propriétés fonctionnelles et non fonctionnelles du service qui représentent sa particularité métier. Elle permet aux consommateurs de décider si ce dernier correspond à leur besoins.

- *propriétés fonctionnelles* : fournissent les informations nécessaires à l'invocation et à la bonne utilisation des fonctionnalités du service. Ces informations peuvent être catégorisées en partie syntaxique et sémantique. La partie syntaxique regroupe les informations strictement nécessaires pour communiquer avec le service (signature, protocole de communication, etc.). La partie sémantique regroupe les informations qui permettent de bien utiliser le service (sémantiques attachées aux fonctions et aux données échangées, comportement extérieur, ordonnancement des invocations aux différentes fonctionnalités, etc.).
- *propriétés non fonctionnelles* : fournissent les informations de qualité de service (QoS, performance, coût, sécurité, etc.) et sur les politiques du fournisseur (localisation, législations d'exploitation du service, utilisation des données clients, etc).

Ces informations sont généralement attachées à des ontologies de domaines [Gru93] pour permettre leur compréhension sémantique automatisée. De fait, les traitements automatisés qui sont les supports clés de la dynamicité reposent sur le formatage correcte de ces informations descriptives. Le rôle central de la description de service sera illustré dans les sections suivantes.

1.2.3.2 Architecture orientée services : AOS

Le SOSE repose sur la notion d'architecture orientée services (AOS [OAS08, PH07]) qui définit les guides pour organiser la construction d'applications par des services. L'AOS introduit les notions de fournisseurs et de consommateurs de services :

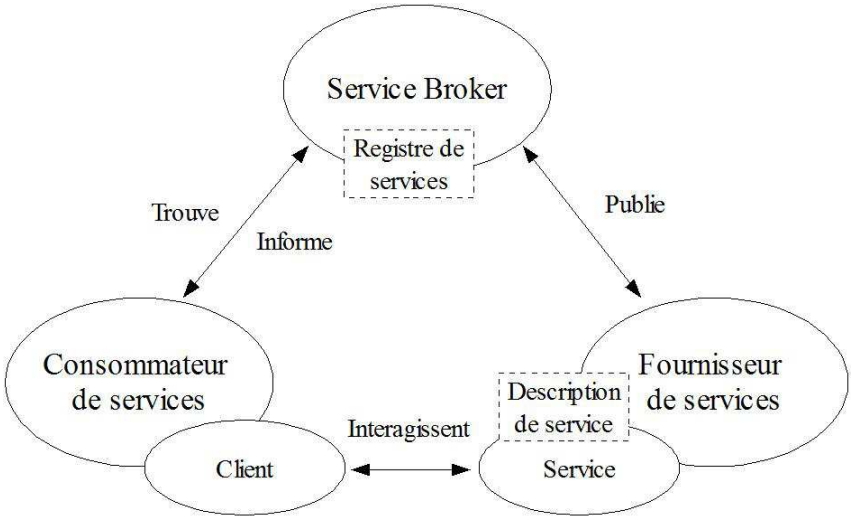

Fig. 1.1 – Organisation d'une architecture orientée services

- le *fournisseur de service* : est l'acteur responsable du développement du service, de son déploiement et son exécution, de sa maintenance en cas de besoin, puis de sa cessation d'activité lorsqu'il est arrivé à son terme.
- le *consommateur de service* : est l'acteur qui utilise des services en fonction de ses besoins.

Fournisseurs et consommateurs sont au départ indépendants, c'est-à-dire que le fournisseur, lors de l'implémentation de son service, n'a pas de connaissances à priori sur ses futurs consommateurs ni sur la façon dont ils réutiliseront son service. Ainsi, l'AOS repose sur un troisième acteur le "*service broker*" [OAS08].

- le "service broker" : est l'acteur associé à un registre de services qui permet de faire le lien entre consommateurs et fournisseurs qui s'ignorent. Les fournisseurs publient leurs services dans ces registres qui sont ensuite consultés par les consommateurs afin de déterminer ceux qui correspondent à leur besoin. Dans la suite de la thèse, le service broker sera directement englobé derrière le terme de registre de services.

Fournisseurs et consommateurs sont ensuite liés par l'utilisation du service. Tous deux s'engagent à respecter un contrat d'utilisation, en terme de respect de l'interface du service pour le consommateur, et de respect des propriétés fonctionnelles et non fonctionnelles promises pour le fournisseur.

La figure 1.1 résume l'organisation d'une architecture orientée services.

Un architecte logiciel qui suit cette organisation se place en tant que consommateur des multiples services qu'il a identifié en fonction de ses besoins. Puis, il définit la collaboration entre ces services pour réaliser son application.

En résumé, le SOSE repose sur :

- le service : qui encapsule de façon homogène des ressources à nature hétérogène ; et
- l'AOS : qui définit les règles de construction d'une application orientée services et les rôles des différents acteurs afin de permettre l'établissement de collaborations

dynamiques via les réutilisations de services existants.

Pour assurer les principes de l'AOS, le SOSE introduit différentes mécaniques opératoires qui sont présentées dans la section suivante.

1.3 Mécaniques opératoires

Un service est développé, puis offert à l'exploitation par un fournisseur dans le but d'être réutilisé par des consommateurs.

Les mécaniques opératoires développées dans le SOSE servent à établir ce lien d'exploitation entre consommateurs et fournisseurs. Ces établissements de liens doivent être manipulés dynamiquement afin de supporter la volatilité des environnements cibles. Pour faciliter ces manipulations dynamiques, le SOSE prône la minisation des dépendances entre services en collaboration, c'est la notion de *couplage faible* [Kay03].

Le SOSE peut être résumé en deux mécanismes principaux :

- la **publication de services** [OAS08, PH07] : sous la responsabilité du fournisseur de services, elle rend visible le service dans l'environnement afin que les clients potentiels soient capables de trouver ce service et donc être au courant de son existence et des fonctionnalités qu'il offre ;
- la **composition de services** [OAS08, PH07, NGM+08, CMMC08] : sous la responsabilité du consommateur de services, elle établit le lien d'exploitation avec le fournisseur de services et combine les différents services réutilisés afin de réaliser l'application. Cette composition regroupe les étapes de découverte des services disponibles qui peuvent répondre aux besoins du consommateur, de sélection des plus adaptés aux préférences, et de définition et gestion des collaborations entre les services effectivement utilisés.

1.3.1 Publication de services

La publication de services est le mécanisme qui permet la mise à disposition des services aux clients potentiels. Le principe est de rendre les descriptions des services visibles aux clients. Ainsi, ils peuvent décider de l'emploi ou non des services associés, puis être capables de communiquer correctement avec eux afin d'en tirer les résultats attendus. Normalement, le fournisseur de services est responsable de la cohérence entre les descriptions de services publiées et son service. À tout instant, il s'engage à garantir les résultats retournés.

Les descriptions des services disponibles sont donc la cible des algorithmes de publication qui les répertorient dans des registres de services tels que [OAS04, VGS+05]. Ces registres représentent les lieux communs, connus par les parties fournisseurs et consommateurs. Ils sont les points d'entrée de ces consommateurs qui sont à la recherche de services particuliers.

Les travaux de recherche autour des registres s'orientent naturellement vers une optimisation des classifications des descriptions de services et leur cohérence avec les

services en exécution. L'approche classique correspond à un regroupement fonctionnel sémantique [VGS+05] basé sur des ontologies [Gru93] et la définition d'une topographie entre ces regroupements, c'est-à-dire la définition de relations entre ces ensembles de descriptions détaillant une certaine logique d'organisation, par exemple l'équivalence entre registres.

Ainsi, les méthodes de publication sont associées à un certain processus de classification et donc à un certain processus de *découverte de services* pour parcourir de façon optimale cette classification de services. La découverte de services correspond à la première étape de l'autre mécanisme central du SOSE : la composition de services.

La publication de services est la préoccupation du fournisseur de services qui cherche à mettre son service à disposition des clients. La composition de services est la préoccupation du consommateur de services qui cherche à réutiliser des services existants pour remplir certains besoins de son application.

1.3.2 Composition de services

La *composition* est le processus de construction d'une nouvelle entité logicielle à partir d'autres entités préexistantes. L'entité résultat est dite *composite*. Ainsi, le processus de composition de services correspond à la spécification d'un *service composite* [OAS08] à partir d'autres services. Le service composite est la réification de la collaboration entre services de manière à appréhender le résultat de cette collaboration comme un service à part entière. Cette approche renforce la réutilisabilité où le service composite peut être manipulé de façon homogène par tout processus SOSE ciblant les services. Elle permet aussi la construction claire de compositions hiérarchiques via des compositions de composites.

Le processus de composition de services peut être divisé en trois étapes, *découverte de services*, *sélection de services* et *composition de services*.

1.3.2.1 Découverte de services

La *découverte de services* est un processus qui identifie les services qui peuvent répondre aux besoins exprimés par l'architecte. Un algorithme de découvertes compare les descriptions de services dont il a accès avec les exigences fonctionnelles et non fonctionnelles qui lui sont fournies. Ces comparaisons déterminent les *services candidats* à l'utilisation, c'est-à-dire les services existants qui, d'après leur description de service, peuvent répondre aux exigences. Pour établir la liste de ces services candidats deux problématiques se posent à un moteur de découvertes : i) sur l'organisation et l'accès aux services disponibles [VGS+05,KKS07], et ii) sur la comparaison entre la description de ces services et les besoins puis la reconnaissance d'une solution comme adéquate [LL09,PC09].

La première problématique est en rapport direct avec la publication de service et les approches classiques de fouille de données sur des registres de descriptions. La seconde problématique fait intervenir des éléments plus spécifiques au SOSE sur lesquels nous allons nous focaliser.

Ainsi, il existe différentes approches d'alignement entre besoins et descriptions des services qui s'organisent suivant deux grandes variables :

- le *type de correspondance* : les algorithmes de découverte peuvent être classés en approches 1-1 et 1-N [KKS07]. L'approche 1-1 [GNY04, VGS+05] correspond à l'identification d'exactement un service existant pour répondre à un besoin particulier. De son côté, une approche 1-N [KKS07, BP08] correspond à l'identification d'une composition de plusieurs services pour supporter ce besoin. Une telle approche est capable de construire des solutions qui n'existent pas directement dans le système en composant jusqu'à N services disponibles. Ainsi, elle doit supporter la spécification des bonnes collaborations entre ces N services en corrélation avec les exigences du consommateur ; et
- le *degrés de pertinence* de la solution : les algorithmes de découverte varient suivant leurs capacités à gérer la pertinence des solutions retournées par rapport aux propriétés fonctionnelles et non fonctionnelles exigées. La prise en compte des sémantiques par l'emploi d'ontologies de domaines d'applications, dans la formulation des besoins et des descriptions de services, améliore significativement cette pertinence [RHL09, MWL+08].
  Dans le cas où les écarts avec les besoins sont toujours significatifs, les solutions les plus proches possibles sont proposées puis laissées aux choix de l'architecte de les utiliser ou non.

1.3.2.2 Sélection et négociation de services

La seconde étape de la composition de services correspond au processus de *sélection de services* qui identifie, parmi les services candidats, la solution de réalisation la plus adaptée aux besoins de l'application en construction. En général, ces services candidats répondent tous aux conditions minimales d'acceptation posées par l'architecte de l'application. La sélection se base donc sur ses préférences "utilisateurs" (en tant que futur client) pour identifier celui avec la plus haute qualité de service relative.

La sélection d'un service s'accompagne d'une phase de *négociation* entre consommateur et fournisseur qui vise à établir un *contrat d'exploitation* [OAS08]. Ce contrat stipule les droits et les devoirs de chacun des participants, et en particulier, il statue des conséquences de son éventuel non respect.

Ainsi, le contrat détermine les propriétés fonctionnelles et non fonctionnelles [ZMCW09, ZM11] qui seront garanties au consommateur si ce dernier se conforme à leurs règles d'emploi. Dans le cas où ces fonctionnalités ne sont plus remplies à cause d'une défaillance avérée côté fournisseur, un certain nombre de sanctions et de compensations peuvent être mises en place jusqu'à la possibilité extrême d'une rupture de la collaboration [OAS08]. Le processus de fin d'exploitation du service doit donc faire partie du contrat, qu'il soit issu de cette rupture définitive ou tout simplement par arrêt décidé à l'avance (temps d'exploitation ou nombre d'utilisations fixé qui a été atteint, fin de renouvellement de contrat, etc.)

La phase de négociation inclut donc toutes les possibilités de variabilité et de

personnalisation où le consommateur peut demander des comportements fonctionnels et non fonctionnels particuliers du service, non prévus par son fournisseur. La gestion de cette variabilité dans l'établissement d'un contrat représente un verrou extrêmement fort dans la recherche qui limite son automatisation.

1.3.2.3 Composition de services

L'étape de composition de services correspond à l'établissement de la collaboration entre les services qui ont été sélectionnés. Le résultat final est la construction de la composition de services, classiquement encapsulée sous la notion de service composite [OAS08, LML+08]. Dans cette thèse, nous nous intéressons particulièrement à cette étape de construction et détaillons de façon plus exhaustive l'état de l'art de son fonctionnement.

1.4 Composition dynamique de services

La composition de services est une des problématiques centrales du SOSE [NGM+08, BHI10, CMMC08]. Il existe dans la littérature un grand nombre de travaux[3], théoriques ou pratiques, qui abordent cette composition et en particulier son automatisation. Afin de replacer ces travaux les uns par rapport aux autres, nous utilisons un scénario très simple issu des environnements Iam[4] [Wei91]. Deux appareils, un jukebox et un microphone, sont des ressources disponibles de l'environnement, et chacune d'elles est encapsulée dans un service. Ces deux services collaborent pour assurer une fonctionnalité de plus haut niveau où un utilisateur dit le nom d'une chanson voulue, le microphone récupère ce choix et le transfère au jukebox qui joue cette chanson si elle se trouve dans son répertoire. Cette composition simple entre deux services pose les trois types de problèmes suivants :

- la **collaboration entre les services** : c'est-à-dire la définition et la gestion des différents flots de contrôle entre les services qui assurent leurs invocations au bon moment en leur fournissant les bonnes données. Dans notre exemple, cette collaboration correspond au microphone qui récupère un son puis transmet l'information au jukebox ;
- la **gestion des hétérogénéités** : c'est-à-dire la capacité à s'abstraire de la nature hétérogène des services, ou comment invoquer et réutiliser un service qui fournit les bonnes fonctionnalités suivant la bonne QoS mais qui ne respecte pas les contraintes de réalisation prévues par l'architecte. Dans notre exemple, le jukebox nécessite un string représentant le nom de la chanson alors que le microphone produit un flux audio lorsqu'il fournit ce nom. Un mécanisme de médiation entre ces incompatibilités est obligatoire pour permettre la compréhension respective de la donnée entre ces services ; et
- l'**adaptation contextuelle** : un dernier problème est d'assurer la flexibilité d'une composition de services. En effet, le SOSE cible des environnements hautement

[3] DBLP : mots clés *service composition* correspondent à près de 2000 articles
[4] Intelligence Ambiante

volatiles. Les compositions doivent donc être capables d'effectuer des modifications dynamiques et contextuelles de leur architecture par la gestion pertinente des ajouts ou des retraits de services [NGM+08, CDA08]. Dans notre exemple, on peut supposer que le jukebox devienne indisponible mais que le système possède deux autres services qui puissent servir de solution alternative : un lecteur MP3 et des enceintes. Le lecteur ferait la sélection de la chanson et les enceintes en diffuseraient le son. L'adaptation contextuelle doit avoir la capacité de détecter les défaillances, d'identifier les solutions alternatives disponibles et de les incorporer en remplacement, tout en garantissant la cohérence globale de la composition et donc son exécution.

1.4.1 Collaboration entre services

La collaboration entre services correspond à l'établissement de différents flots de contrôle distingués en *flots de travail* et *de données* entre les services préalablement sélectionnés pour réaliser des fonctionnalités plus complexes :

- *flots de travail* : détermine l'ordre des invocations des différents services, les différents embranchements de l'exécution, les séquences parallèles ou continues, etc.
- *flots de données* : détermine les échanges de données entre les différents services.

De nombreux travaux existent sur la modélisation, la formalisation et l'exécution des différents flots de contrôle. Mais, dans la problématique SOSE, on parle typiquement d'organisation de ces flots suivant deux types de schémas de collaboration, l'*orchestration* et la *chorégraphie* [DP06].

1.4.1.1 Orchestration et chorégraphie

Différentes définitions existent de l'orchestration et de la chorégraphie, proposées par des recherches individuelles [ACKM03, C03] ou dans le cadre d'organismes de normalisation [OAS08, W3C06]. Sans entrer dans une liste des définitions et de leur différenciations, nous pouvons définir les noyaux conceptuels qui spécifient l'orchestration et la chorégraphie.

L'**orchestration** correspond à un schéma de collaboration centralisée, où une seule entité réunit l'ensemble des appels aux différents services de la composition (figure 1.2). Cette entité invoque les services dans un ordre prédéfini. Elle se charge de la récupération des différents résultats et du transfert des données pertinentes pour l'exécution correcte des services invoqués. Typiquement, l'entité centralisante est associée à la notion de service composite, et les services invoqués sont les constituants de ce composite. L'orchestration peut être vue comme une composition verticale où les communications existent uniquement entre composite et constituants.

La **chorégraphie** correspond à un schéma de collaboration distribuée où les échanges de messages se font directement entre les services en coopération (figure 1.2) afin d'effectuer une tâche particulière vers un but défini. Typiquement, cette coopération

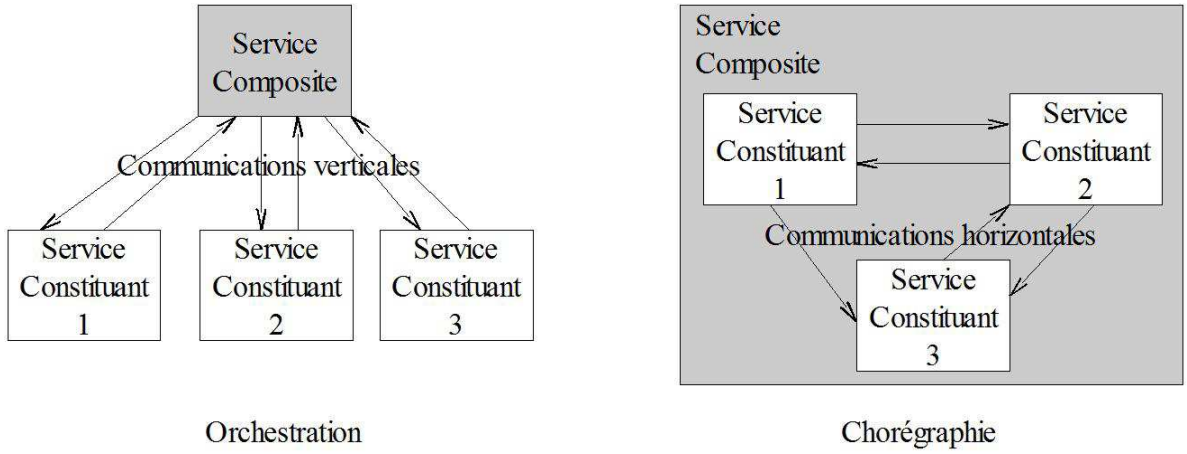

Fig. 1.2 – Orchestration vs Chorégraphie de services

correspond à une composition horizontale où les communications sont exclusivement entre services constituants.

Orchestration et chorégraphie correspondent donc à deux types d'approches différentes de spécification de la collaboration qui organisent les communications entre les services. Pour illustrer tous les aspects de gestions des collaborations et des communications, nous prenons l'exemple de la composition entre services Web [RS04, DS05].

1.4.1.2 Problématique de la collaboration : l'exemple des services Web

Dans les technologies services Web, l'*orchestration* est typiquement manipulée dans une *perspective privée* [DP06], c'est-à-dire que le service composite, qui centralise les invocations aux services constituants, est une boîte noire. Ainsi, les clients de ce composite ignorent son fonctionnement interne, de même que les services constituants ignorent leur rôle dans la composition ainsi que les autres participants. La *chorégraphie* adopte, de son côté, une *perspective publique* [DP06] où les échanges de messages entre les différents services constituants sont directement visibles de l'extérieur. Chaque participant sait exactement son rôle et le rôle des autres dans la collaboration. Le service composite qui encapsule une chorégraphie de services est une boîte transparente sans réelle existance d'un point de vue architectural.

La figure 1.3 fait un récapitulatif des technologies les plus répandues dans la composition de services web, standardisées par OASIS[5] et W3C[6]. Ce récapitulatif est découpé suivant les perspectives publiques et privées, que l'on se place respectivement dans une approche par chorégraphie ou par orchestration.

Les différents standards sont organisés suivant deux axes : i) les orientations choisies et ii) les préoccupations qu'ils ciblent.

Nous identifions trois orientations principales :

[5]http ://www.oasis-open.org/
[6]http ://www.w3.org/

- *orientée exécution* : approche classique de composition de services web qui se concentre sur la compatibilité syntaxique ;
- *orientée but sémantique* : approche de composition qui prend en compte les sémantiques liées aux ontologies de domaines pour améliorer les différents processus de publication, découverte, sélection et composition de services ; et
- *orientée qualité* : approche de composition qui repose sur des descriptions de services particulières pour maximiser la qualité de la composition.

Les technologies sont ensuite réparties en huit préoccupations, en fonction du niveau sur lequel elles interviennent dans l'établissement d'une collaboration entre services web (figure 1.3) :

- *transport* : regroupe les protocoles de transport issus de la couche application (7e couche du modèle OSI [Zim80]) responsables des transferts d'informations. Les services web utilisent HTTP, RCP, SMTP, IIOP etc. [W3C11a, Rho99, SR09, ACKM03].
- *message* : regroupe les technologies de formatage de messages pour la spécification des échanges d'informations structurées. Elles reposent sur les protocoles de la couche application précédente pour la transmission de messages. Le SOAP [SR09, ACKM03] est la technologie privilégiée utilisée par les WS[7].
- *description de services* : regroupe les technologies de description de services web pour décrire leurs fonctionnalités offertes et leurs conditions d'utilisation. WSDL [W3C11b] est le standard de description le plus utilisé dont certaines extensions gèrent les ontologies [Gru93]. D'autres technologies sont plus spécifiquement orientées, telles que OWL-S [W3C04a] pour les ontologies et MAIS SDL [MMMP04] pour la qualité.
- *registres des services* : regroupe les technologies pour classifier et répertorier les descriptions de services directement liées aux processus de publication et de découverte de services. UDDI [OAS04] est la technologie la plus connue.
- *coordination des communications* : regroupe les technologies exécutables pour coordonner les différents envois de messages. Les technologies dominantes sont les propositions BPEL [OAS07] pour l'orchestration et WS-CDL [W3C04b] pour la chorégraphie.
- *sémantiques de description* : regroupe les technologies pour l'expression des sémantiques via des ontologies de domaine où OWL-S et WSMO [RKL+05] sont des représentants importants.
- *description de composition* : regroupe les technologies pour la description d'un service composite et de sa coordination des services constituants. La chorégraphie étant typiquement publique, l'encapsulation boîte noire n'existe réellement en WS qu'à travers l'orchestration et les technologies BPEL, BPML ou BPMN [OMG11].
- *accords commerciaux* : regroupe les technologies pour le formatage de contrats commerciaux entre participants où SLA [LKD+03], qui exprime leurs obligations qualitatives, est un des représentants principaux.

[7] Web service : service Web

	Perspective publique: Chorégraphie		Perspective privée: Orchestration				
	Orientée exécution		Orientée exécution		Orientée but sémantique		Orientée qualité
	OASIS	W3C	W3C	OASIS	W3C DAML	DERI	MAIS
Accord commerciaux	ebXML CPA	SLA (IBM)					
Description de composition			BPML BPMN (OGM)	BPEL	OWL Reasoner	IRS-III Reasoner	WSMO
Sémantiques de description					OWL-S	WSMO	
Coordination des communications	ebXML BPSS	WS-CDL	WSCI	BPEL			BPEL
Registres des services	ebXML Rep.	UDDI			IRS-III		URBE
Description de services	ebXML CPP	WSDL			OWL-S		MAIS SDL
					WSDL		WSDL
Message	SOAP						
Transport	HTTP, RCP, SMTP, IIOP, ...						

Fig. 1.3 – Technologies de collaboration de services Web

Ainsi, la figure 1.3 propose un échantillon des technologies les plus représentatives du service Web et permet de mieux comprendre les différents niveaux à considérer. Elle met en valeur la complexité de la problématique de collaboration entre services à travers l'ensemble des préoccupations et orientations qui doivent être prises en compte.

Il existe un nombre très important d'autres propositions telles que les dérivés WS-* [DP06] (WS-security, WS-transaction, etc.), cependant les répertorier se trouve hors du cadre de ce travail. En effet, pour développer notre approche, nous nous intéressons plus particulièrement aux niveaux *coordination des communications* et *description de composition*. Dans la section suivant, une répartition de références illustratives des préoccupations issues des écoles européennes, américaines et asiatiques, et françaises met en perspective les concepts importants de ces deux niveaux.

1.4.1.3 Références illustratives

École européenne :

Dans [Wom09], l'auteur étudie les dépendances entre la chorégraphie qui gére les collaborations publiques des différents participants et leur orchestration interne qui dirige leur fonctionnement privé individuel. En particulier, il propose une approche pour propager de façon cohérente les changements effectués par un participant au niveau de la chorégraphie sur les orchestrations des autres participants. Cette approche repose sur une méthode formelle pour représenter sémantiquement les chorégraphies et la nature de leurs changements. Ces changements sont ensuite projetés sur les représentations sémantiques

des orchestrations. Enfin, ces représentations sémantiques sont à leur tour projetées sur leurs représentations syntaxiques qui correspondent aux processus d'orchestration exécutables.

Dans [DR09], les auteurs étudient aussi ces dépendances entre chorégraphie et orchestration mais sous une autre perspective. Ils proposent un ensemble de méthodes formelles pour déduire les orchestrations internes des participants à partir de leur collaboration externe modélisée par une chorégraphie. Pour remplir cet objectif, ils identifient les caractéristiques importantes des orchestrations et des chorégraphies puis les expriment à travers leurs propres langages formelles. À partir d'un ensemble de règles, ils dérivent de la chorégraphie d'entrée les possibilités d'orchestrations pour chaque participant.

École américaine et asiatique :

Dans [CK07], les auteurs analysent les points de variabilité sur la collaboration qui peuvent exister dans une composition de services. Ils catégorisent ces points de variabilité en trois types : variabilité sur les flots de contrôles, variabilité sur les services constituants utilisés et variabilité sur les logiques métiers de ces services. Enfin, ils proposent un processus de développement en cinq phases pour construire des compositions adaptables qui prennent en compte ces informations de variabilités.

Dans [FFL10], les auteurs se concentrent sur la relation entre le modèle de description d'un processus d'orchestration et ses instances en cours d'exécution. Lors de modifications sur le modèle, seules les instances créées après ces modifications sont conformes au nouveau processus alors que les anciennes instances, en cours d'exécution, ne pourront être modifiées qu'à partir de leur arrêt. Cet arrêt est extrêmement problématique dans le cas d'exécution de processus de longue durée. Ainsi, les auteurs introduisent une notion de migration continue des instances en exécution. Ces migrations continues sont elles même définies comme des processus de longue durée et reposent sur l'identification de points de migration dans les instances du modèle source vers le modèle modifié.

École française :

Dans [FYG09], les auteurs décrivent une méthodologie flexible pour diviser la spécification d'un processus centralisé de collaboration de services Web de manière à supporter les exécutions distribuées. Les flots de travail et de données sont répartis entre différentes partitions suivant des interactions pair à pair entre ces partitions et via des communications asynchrones. Ce découpage repose sur l'étude des dépendances entre les activités du processus qui sont par la suite manipulés dans une table des dépendances transitives. Leur approche supporte des schémas avancés tels que les boucles ou les instances multiples.

Dans [RF11], les auteurs adoptent une perspective différente sur le processus de composition de services qui est vue comme un artefact ayant son propre cycle vie. Ainsi, ils expriment les collaborations entre services sur quatre niveaux d'abstraction : le niveau abstrait qui décrit une composition en termes de buts à réaliser, le niveau orchestré qui décrit une composition en termes d'activités ordonnancées, le niveau exécutable qui décrit une composition sous la forme d'un processus exécutable, et le niveau trace qui décrit une composition en termes de services effectivement invoqués. À partir de ce découpage, ils

introduisent une notion de patron qui associe un but et un contexte à chaque composition. Ces patrons sont ensuite réutilisables et adaptables.

1.4.2 Gestion des hétérogénéités

Les hétérogénéités des services représentent l'ensemble des variabilités possibles sur les ressources encapsulées (le comportement extérieur des services, leurs implémentations et ainsi de suite). Sans fournir la liste exhaustive de ces hétérogénéités, nous pouvons cependant les regrouper en deux catégories :

- *hétérogénéités de sémantique métier* : elles regroupent les éléments liés au résultat attendu d'une exploitation d'un service en terme fonctionnel et non fonctionnel. Elles spécifient les fonctionnalités fournies par le service, sa qualité de service (QoS, sécurité, fiabilité, etc.). De plus, cette catégorie regroupe tout type d'informations additionnelles tel que la localisation du fournisseur, la langue de travail, etc.
- *hétérogénéités de réalisation technique* : elles représentent les implications techniques pour remplir les fonctionnalités attendues. Ces hétérogénéités peuvent être réparties en trois groupes : les *aspects technologiques*, les *aspects conversationnels*, et les *aspects données*.

1.4.2.1 Aspects technologiques, conversationnels, et de données

L'objectif de la gestion des hétérogénéités est de permettre les collaborations entre entités à première vue incompatibles. Dans le cas du SOSE, c'est faire correspondre les besoins requis par l'architecte avec certains services existants. Les besoins fonctionnels et non fonctionnels sont directement liés à la sémantique métier. L'adaptation d'un comportement fonctionnel d'un service est directement sous le contrôle de son fournisseur. Typiquement, ces adaptations de sémantique métier ne se font pas par gestion des hétérogénéités mais plutôt par le processus de *négociation* (section 1.3.2.2) entre le futur consommateur et ce fournisseur.

Le domaine d'actions de la gestion des hétérogénéités concerne principalement les hétérogénéités de *réalisation technique*. Sa finalité est de permettre l'utilisation de tous services qui répondent aux besoins fonctionnels et non fonctionnels de l'architecte quelque soit les techniques employées pour leur réalisation. Elle est le garant des propriétés d'interopérabilité entre services et d'indépendance de protocoles [Erl05, Jos07, BL07].

Les travaux autour de la gestion des hétérogénéités se focalisent principalement sur trois aspects :

- *aspects technologiques* : cette catégorie regroupe les problèmes de compatibilité relatifs aux technologies utilisées telles que les protocoles de transports de messages, les formats et cryptages de données, les technologies d'implémentations, etc. Cette problématique de passerelle entre technologies est bien plus ancienne que le SOSE qui ne fait qu'appuyer à nouveau ce besoin en cherchant l'abstraction technologique [NGM+08].

– *aspects conversationnels* : les aspects conversationnels font directement référence aux problèmes d'interopérabilité entre services en collaboration qui ont des comportements extérieurs incompatibles [CNP09]. De fait, un service impose une succession précise d'appels de ses fonctions pour l'utiliser de façon correcte. L'établissement du schéma de conversation approprié entre le service et son client est donc nécessaire.
– *aspects données échangées* : cette catégorie se focalise sur la compatibilité des données échangées entre services en collaboration, c'est-à-dire que l'ensemble des services d'une même composition interprètent les données de la même façon, et que le sens qui y est attaché est conservé de l'un à l'autre [CRZ09, PY10]. Par exemple, un integer qui représente une date peut être au format anglophone (mois, jour puis année) ou francophone (jour, mois puis année), ou encore une valeur monétaire exprimée en euros ou en dollars.

Différents travaux existent et abordent ces trois aspects hétérogènes de façon isolée ou dans leur ensemble. Typiquement, la gestion des hétérogénéités est manipulée suivant deux visions distinctes : soit par le biais d'une entité architecturale dédiée telle que les médiateurs ou les proxy [CNP09], soit par son incorporation statique dans une plate-forme cible [OAS09].

1.4.2.2 Références illustratives

École européenne :

WSMO [RKL+05] est une approche de modélisation qui vise la gestion des hétérogénéités en général. Tout d'abord, il sert à la description des services web sémantiques [MSZ01], c'est-à-dire des services web combinés à des ontologies. Il exprime les contraintes de coordinations via des sémantiques pour les enjeux de composition. Ces contraintes servent ensuite à résoudre les problèmes d'intégration. Son approche repose sur la définition de médiateurs comme entités de premier ordre responsables de l'interopérabilité entre contraintes de services a priori incompatibles.

La définition d'entités intermédiaires est une approche classique de la gestion des hétérogénéités. Par exemple les articles [CDN08, CRZ09] et [CNP09], qui abordent respectivement les aspects données échangées et conversationnels, se basent sur une notion de *proxy* entre l'entité consommatrice et le service qu'elle réutilise. Ces proxy interprètent au runtime des fichiers scripts générés dynamiquement à partir des contraintes de communication posées par les deux parties. Ces générations de scripts permettent les changements dynamiques où un même proxy peut être réutilisé pour invoquer d'autres services.

École américaine et asiatique :

Dans [KKS07], les auteurs proposent une méthode de composition de services qui cible les problèmes d'hétérogénéités de données. Leur approche traite ce problème en amont, au niveau des processus de publications et de découvertes de services, en se basant sur une organisation particulière des services disponibles. De fait, les services sont organisés dans un graphe qui représente l'ensemble des compositions possibles. Lors d'incompatibilités

de données entre les services, le système utilise le graphe des compositions pour identifier une succession de services capable d'assurer les modifications nécessaires sur la donnée à priori incompatible.

Dans [JWY+10], les auteurs s'intéressent aux hétérogénéités des aspects conversationnels et de données. Ils étudient les incompatibilités des ontologies attachées aux interfaces des services réutilisés. Ils proposent de résoudre ces incompatibilités par une méthode d'extraction de ce qu'ils nomment les sous-ontologies ("*sub-ontologies*" ou *Representation ontology*) qui sont des expressions simplifiées des ontologies d'origine. L'emploi de ces sous-ontologies permet de simplifier le processus d'alignement d'ontologies et d'augmenter sa précision. À partir de ce travail d'extraction et d'alignement, les auteurs sont capables de générer dynamiquement des adaptateurs d'interface pour garantir les communications.

École française :

Les équipes INRIA sont très actifs sur l'aspect technologique de la gestion des hétérogénéités, en particulier dans l'extension d'une technologie existante appelée SCA (Service Component Architecture [OAS09]). Des plate-formes Eclipse [Ecl09] permettent le développement d'applications orientées services à base d'assemblages de composants SCA. L'atout majeur de SCA est de supporter les variabilités dans l'implémentation des composants SCA ou de leurs communications qui peuvent être réalisés par de nombreuses technologies différentes (Java, C++, WSDL, RMI, etc.). L'implémentation INRIA de SCA, appelée FraSCAti [SMF+09], améliore le nombre de technologies en supportant REST [Fie00], UPnP[8], OSGi [OSG11], etc. Ainsi, les différents éléments de l'architecture SCA peuvent être réalisés dans des technologies hétérogénes, par la suite la plate-forme se charge des communications transparentes entre ces éléments.

Plus de détails sur les apports de SCA FraSCAti de l'INRIA par rapport à SCA d'origine sont donnés dans le chapitre 5 sur l'implémentation et dans l'annexe A.

Dans [BP08, PY10], les auteurs se focalisent sur les aspects conversationnels et de données. Ils proposent des méthodes basées sur la combinaison de techniques de plannification de graphes [CBB07, GNT04] avec des descriptions sémantiques attachées aux données et aux fonctionnalités qui composent les comportements. Ces méthodes supportent ce que les auteurs appellent les adaptations horizontales (sur les données échangées entre services) et verticales (les incompatibilités entre les besoins et les fonctionnalités).

1.4.3 Adaptation contextuelle d'une composition de services

Le SOSE cible le développement d'applications devant supporter des environnements volatiles. Les compositions de services qui implémentent ces applications doivent donc être suffisamment flexibles pour suivre les évolutions de ces environnements et ainsi évoluer à leur tour. Les modifications dynamiques d'une composition de services suivent les principes clés de l'auto-adaptation [BCDW04, Bra04, Hof93] en particulier la notion

[8]Universal Plug and Play

d'*efficacité* ("*effectiveness*"), c'est-à-dire l'adéquation entre le besoin réel d'adaptation, son coût et les bénéfices attendus.

1.4.3.1 Fonctionnement global

D'une manière générale, nous représentons une composition de services comme un graphe avec des noeuds (les services) et des arcs (les relations entre ces services). Les évolutions sur ce graphe sont classées suivant leur cible : le noeud, l'arc, ou le graphe.

- *évolutions sur les noeuds* : du point de vue d'un service composite, les services constituants qu'il réutilise sont des boîtes noires appartenant à des fournisseurs tiers. Ils ne peuvent donc pas être la cible d'évolutions contrôlées par le service composite. En fonction de ses besoins, ce dernier peut uniquement re-négocier son contrat d'utilisation avec les fournisseurs pour leur demander des changements sur les fonctionnalités qu'il reçoit.
- *évolutions sur les arcs* : les relations entre services sont figées par les sémantiques liées à la composition de services et aux différentes flots de données et de travail (section 1.4.1). Les évolutions des arcs sont hors du contrôle du service composite.
- *évolutions sur le graphe* : l'organisation des différents flots et le choix des services qui participent à sa composition sont sous le contrôle du service composite. Ainsi, l'ensemble des méthodes d'adaptations contextuelles d'un service composite se focalise uniquement sur des modifications du graphe en lui-même, c'est-à-dire par ajouts ou retraits de noeuds ou d'arcs qui représentent les ajouts ou les retraits de services constituants et la modification des flots de contrôle en conséquence.

L'adaptation contextuelle d'une composition de service peut se découper en deux grandes étapes :

- l'*observation contextuelle* : la composition doit être capable d'observer son environnement, c'est-à-dire de récupérer et d'identifier les informations pertinentes sur les différents contextes clients ou systèmes. Concrètement, le fournisseur de la composition doit identifier tout décalage entre besoins et réalisation actuelle. Ces besoins sont ceux de ses clients qui demandent de nouvelles fonctionnalités que sa composition actuelle ne peut remplir. Ils sont aussi ses propres besoins qui ne sont plus en adéquation avec les services constituants qu'il réutilise. Ainsi, ces pertes d'alignements sont dues : soit aux changements de besoins du côté de la composition, soit aux services réutilisés qui ne sont plus capables d'assurer le contrat d'utilisation préalablement établi.
- l'*identification et l'application de la solution adaptée* : à partir des besoins précédents, le service composite doit identifier la nouvelle composition de services qui puisse y correspondre. Il doit déterminer les implications en termes de retraits et d'ajouts de noeuds et d'arcs et mettre en place cette nouvelle solution. Il doit conserver la cohérence de l'ensemble de la composition vis à vis des contextes d'exécution des clients et aussi garantir la continuité de service.

1.4.3.2 Références illustratives

École européenne :

Dans [BGPT09], les auteurs proposent l'intégration de deux modèles, Dynamo [BGP07] et Astro [MPT08], pour supporter la surveillance de processus d'orchestration BPEL en exécution afin de détecter d'éventuelles violations de contraintes prédéterminées. Les techniques issues des aspects[9] [KLM+97] développées dans Dynamo sont utilisées pour récupérer des données fonctionnelles et non fonctionnelles sur l'exécution d'un processus BPEL. Ces données sont ensuite traitées par des modules logiciels, appelés *moniteur*, développés dans Astro. Un moniteur est généré à chaque création d'un processus et dédié exclusivement à celui-ci. Il s'assure de la cohérence des données observées avec les contraintes d'exécution pré-établies.

Dans [CCG+09], les auteurs se focalisent sur l'auto-adaptation d'une composition de services pour respecter et garantir des exigences de qualité de services. Ils présentent l'approche MOSES dont l'idée directrice consiste à ne pas seulement utiliser l'unique service sélectionné pour répondre à un besoin particulier de l'application mais de faire la liaison avec l'ensemble des services candidats identifiés par le processus de découverte de services. À partir de là, la question n'est plus quels services utilisés mais comment utilisés les services candidats pour assurer à tout instant les contraintes de qualité.

École américaine et asiatique :

Dans [YS07], les auteurs étendent la technologie BPEL pour supporter les adaptations dynamiques en fonction des contextes. Ils introduisent des informations sémantiques par l'emploi de OWL-S [W3C04a] et proposent leur propre formalisme pour les descriptions des services disponibles et des données contextuelles. Ils utilisent ce même formalisme pour l'expression des besoins de l'application appelés patron de services ("*service template*"). À partir de cette formalisation, ils développent un moteur de composition qui calcule, lors des modifications de contextes, un degrès de correspondance entre patron de services et services disponibles. Lorsque ce degrès indique un décalage trop important entre patron et service utilisé, le système déclenche la recherche d'un service alternatif dépassant la valeur seuil d'acceptation.

Dans [LML+08], les auteurs proposent un cadre de composition dynamique de services basé sur les découvertes qui utilise une approche CNP (Contract Net Protocol [Smi80]). Dans un premier, ils développent une modélisation des notions de services composites et services constituants par des diagrammes d'états transitions. À partir de cette modélisation, ils utilisent leur extension de CNP (ECNP, Extended Contract Net Protocol) pour identifier les meilleures services disponibles, les lier au service composite et coordonner leurs invocations. Enfin, leur système supporte les changements dynamiques de services constituants par d'autres services candidats en cas de dysfonctionnements.

École française :

Dans [ZPG10], les auteurs présentent DISC (Declarative Integrated Self-healing services web Composition), un cadre déclaratif orienté évènements qui unifie dans une seule approche les processus de design, de vérification et de surveillance d'une composition

[9] AOP : Aspect-Oriented Programming, la programmation orientée aspects

de services web. Pour supporter le design d'une composition, il intégre des aspects variés tels que les relations et les contraintes de données, l'établissement dynamique des liaisons avec les services web, les exigences temporelles et de sécurité, et ainsi de suite. À partir de là, il permet l'instantiation, la vérification et l'exécution de cette composition. Lors de cette exécution, la composition est sous une surveillance capable d'identifier des violations déterminées et de calculer un ensemble d'actions de rétablissement. Le choix de raisonner sur des évènements pour la modélisation et la surveillance d'une composition permet d'intégrer d'autres travaux sur la vérification [FRG08], la gestion des politiques d'autorisation [GZCG10], etc.

Dans [HSD10], les auteurs présentent CEVICHE (Complex EVent processIng for Context-adaptive processes in pervasive and Heterogeneous Environments) un cadre pour le support de processus d'orchestration réactifs aux contextes. Son but est de permettre les adaptations de processus à l'exécution en fonction des évolutions des besoins. CEVICHE se base sur CEP (Complexe Event Processing [Luc02]) pour la récupération des informations pertinentes qui décident des adaptations. Ils introduisent SBPL (Standard Business Process Language), une extension de BPEL, qui permet aux utilisateurs la définition de points d'adaptation dans le processus. Enfin, CEVICHE supporte l'injection dynamique, suivant ces points d'adaptation, de processus alternatives basés sur une approche composant.

En résumé, le processus de composition dynamique de services est extrêmement complexe et pose un grand nombre de problématiques différentes, réparties suivant les trois axes précédents :

- la définition et la gestion des collaborations entre services réutilisés ;
- la gestion des hétérogénéités possibles entre ces services ; et
- la flexibilité de la composition par la gestion des adaptations dynamiques.

Nous avons détaillé un simple échantillon représentatif (résumé dans la table 1.1) d'une partie des préoccupations de la communauté "service". De fait, de nombreux autres travaux existent pour chacun de ces axes. Des initiatives de grandes échelles telles que le projet européen SeCSE[10] [The08] montrent les enjeux stratégiques liés aux problématiques du SOSE. À travers cet état de l'art, nous présentons uniquement certains aspects importants du processus de composition dynamique de services. Une classification et une comparaison exhaustive des techniques de résolution de ces aspects est hors du cadre de cette thèse.

1.4.4 Vers notre approche de méta-modélisation d'un service composite

Les travaux existants autour de la composition de services se concentrent sur différentes propriétés, adoptent différents langages ou technologies, travaillent à différents niveaux

[10]Service Centric Systems Engineering, 15,2 millions d'euros co-financé par l'Union Européenne, 17 partenaires académiques et industriels venant de 8 pays différents, débuté le 1er septembre 2004 pour une durée de 48 mois

Tab. 1.1 – Écoles européennes, américaines et asiatiques, et françaises

	Gestion de collaboration		**Gestion des hétérogénéités**			**Gestion des adaptations**
	Orchestration	Chorégraphie	Aspects techno-logiques	Aspects conversa-tionnels	Aspects données échangées	
École européenne						
[Wom09]	X	X				
[DR09]	X	X				
[RKL+05]			X	X	X	
[CNP09]				X		
[CRZ09]					X	
[BGPT09]	X					X
[CCG+09]	X					X
École américaine et asiatique						
[CK07]	X	X				
[FFL10]	X					
[KKS07]					X	
[JWY+10]				X	X	
[YS07]	X					X
[LML+08]	X					X
École française						
[FYG09]	X	X				
[RF11]	X					
[SMF+09]			X			
[PY10]				X	X	
[ZPG10]	X					X
[HSD10]	X					X

d'abstraction et supposent différentes perspectives et différentes applications dans des environnements particuliers. De notre point de vue, cette diversité n'est pas une limitation des solutions courantes mais elle est intrinsèquement liée à la nature du problème. De fait, chacun des travaux présentés précédemment abordent des problématiques particulières qui doivent être prises en compte pour supporter la complexité du processus de composition.

1.4.4.1 Offrir une meilleure compréhension de la composition

Au lieu de travailler sur un aspect particulier de la composition en lui cherchant une solution "ultime", nous essayons de nous focaliser sur certains éléments de leurs collaborations et leurs interdépendances et proposons d'abstraire et de coordonner ces concepts dans un méta-modèle de service composite présenté dans le chapitre suivant.

Ce méta-modèle a pour objectif de réifier dans une seule approche les concepts de

gestion des collaborations, des hétérogénéités et des adaptations dynamiques. Il étudie les travaux existants et cherche à en extraire les principes et à définir leurs dépendances pour apporter une réponse plus globale à la problématique de composition de services. Ainsi, il permet de mieux situer les caractéristiques et les propriétés attendues d'une composition et il clarifie les visions morcellées et souvent dépendantes de technologies de chacun des travaux précédents. L'objectif est d'apporter une meilleure compréhension des enjeux du mécanisme de composition de services. Le principe de coordination des efforts est ce que nous appelons l'*auto-composition*.

Notre méta-modèle est un premier pas vers une définition théorique globale de la notion de service composite et une participation au besoin de consensus sur ces définitions absent du SOSE [ES08]. En effet, les spécifications formelles existantes (listées dans [ES08]), telles que celles fournies par l'OASIS [OAS08] ou W3C [W3C06], apportent une définition incomplète du composite car se focalisant uniquement sur les notions de flots de contrôle entre les services individuels. D'une part, leurs définitions omettent d'inclure les apports des travaux de recherche actuels sur la gestion des adaptations et des hétérogénéités. D'autre part, elles n'abordent pas les problématiques additionnelles qui peuvent se poser pour permettre l'encapsulation d'une composition dans un service composite.

1.4.4.2 Problématiques additionnelles à la notion de service composite

L'encapsulation d'une composition de services en un service composite ajoute deux autres problèmes :

- la *définition de la description de service composite* : un service composite est un service à part entière. Il doit donc être associé à une description de service afin de supporter l'ensemble des processus SOSE. La composition dynamique de services implique idéalement la génération dynamique de la description du service composite à partir des descriptions des services constituants, et l'adaptation de cette description en fonction des adaptations contextuelles de la composition sous jacente. À notre connaissance, cette problématique reste relativement peu étudiée.
- la *gestion du multitenant* : un service est obligatoirement multitenant, le service composite doit être à son tour multitenant. Il doit donc être capable de gérer les différents contextes clients en parallèle.
  D'une manière générale, la problématique de création d'un service "*stateful*" multitenant est encore fortement statique. Lors de son implémentation, le fournisseur du service développe en interne sa gestion des contextes clients [WB09]. Il est aussi en charge du développement des différents comportements de son service négociés avec des clients particuliers.
  Ainsi, un service composite multitenant doit pouvoir gérer dynamiquement ces comportements particuliers où chacun de ces comportements peut correspondre à une composition de services différente.
  À notre connaissance, cette problématique de prise en compte du multitenant dans un service composite qui encapsule des compositions dynamiques de services est nouvelle et peu référencée.

Ainsi, notre méta-modèle de service composite, en plus d'offrir une meilleure compréhension des aspects de composition dynamiques actuelles via l'auto-composition, s'intéresse à la problématique de gestion de multitenant.

Le choix des aspects pris en compte dans notre méta-modèle de service composite ainsi que la définition de l'auto-composition (c'est-à-dire la coordination des aspects sélectionnés) sont restreints par la contrainte de minimiser le plus possible les dépendances à l'intérieur du service composite. Ce principe d'amélioration du couplage faible entre services représente le fil conducteur qui dirige nos propositions.

1.5 Conclusion

Dans ce chapitre, nous avons explicité le fonctionnement global du paradigme service à travers les préoccupations qui ont dirigé sa définition. Nous avons décrit la notion de service et d'architecture orientée services. Nous avons résumé les supports méthodologiques et techniques qui soutiennent un ensemble de principes guides incluant le couplage faible entre les services, les services comme entités boîtes noires, indépendantes et auto-contenues, l'interopérabilité entre services et l'indépendance de protocoles, et la composition dynamique de services [Erl05, Jos07, Kay03].

Nous nous sommes particulièrement intéressés au processus de composition qui est le support principal de la construction d'applications à base de services. Nous avons identifié les différents aspects de cette composition en sélectionnant un panel d'approches significatif qui exprime toute la complexité et la diversité des problèmes rencontrés dans la réalisation des objectifs du paradigme dont la cible est le support aux environnements hautement volatiles et hétérogènes. Ce travail sur l'état de l'art nous permet d'identifier les limites des approches actuelles qui sont les bases du développement de notre proposition d'un méta-modèle de service composite.

Chapitre 2

Composition de services et méta-modèle de service composite

2.1 Introduction

La composition dynamique de services (section 1.4) est un des mécanismes centraux des architectures orientées services (AOS [OAS08, PH07]). Ce mécanisme est le principal support de la réutilisabilité. Il définit les concepts nécessaires à l'exploitation des ressources existantes exposées en tant que services. Il permet la collaboration de ces ressources afin de réaliser des fonctionnalités plus complexes et ainsi construire de nouvelles applications. Dans le cadre de cette thèse, nous nous intéressons particulièrement à la notion de service composite [OAS08, LML+08] qui est indissociable de la composition de services.

Le service composite correspond à la réification d'une composition de services comme un service à part entière. Via son encapsulation dans un composite, la composition est plus facilement réutilisable et composable. En effet, elle devient accessible de façon identique à un service classique. Ainsi, l'ensemble des mécanismes existants qui ciblent les services peuvent être utilisés. De là, la composition et ses fonctionnalités peuvent être mises à disposition d'utilisateurs potentiels par le processus de publication de services. Ces utilisateurs pourront s'appuyer à nouveau sur le processus de composition de services pour exploiter et combiner ses ressources composites. D'autres processus prometteurs tels que la réplication de services ou la migration d'états d'internes lors de remplacements de

services “stateful” (à états) sont potentiellement réutilisables [GMS09, ARK08].

Grâce aux possibilités de constructions incrémentales, la notion de service composite permet l'établissement clair des niveaux de descriptions où chaque service est tour à tour dit composite ou constituant. L'organisation hiérarchique en composition de composites est facilement compréhensible et manipulable.

Dans ce chapitre, nous cherchons à définir un méta-modèle de service composite. L'objectif majeur de ce méta-modèle est d'offrir une meilleure compréhension des concepts et mécanismes impliqués dans le processus de composition de services. Le principe est de réifier au niveau architectural une sélection des concepts importants liés à la composition définis dans la littérature. Ces éléments ont été préalablement identifiés dans le chapitre 1 et répartis suivant leurs trois préoccupations :

- (a) la gestion des collaborations : regroupe l'ensemble des concepts et mécanismes nécessaires à la coordination des services participant à une même composition de services [FYG09, Wom09, CK07, DP06].
- (b) la gestion des hétérogénéités : regroupe l'ensemble des concepts et mécanismes nécessaires à la communication avec les services participant à la composition. Ces éléments de communication spécifient comment invoquer un service afin de pouvoir exploiter ses fonctionnalités offertes de façon correcte [BP08, PY10, CDN08, CRZ09, OAS09].
- (c) la gestion du contexte et des adaptations : regroupe l'ensemble des concepts et mécanismes nécessaires à l'observation des contextes utilisateurs ou systèmes, à l'identification des défaillances et des possibles solutions alternatives [CCG+09, BGPT09, NGM+08, Pap08, ZPG10].

De plus, la réification d'une composition en un service à part entière ajoute deux problèmatiques spécifiques relativement peu étudiées (section 1.4.4.2) :

- (d) la gestion du multitenant : un service est multitenant [Jac05], le service composite doit à son tour pouvoir gérer de multiples contextes clients.
- (e) la génération d'une description de service composite : pour être traité de façon homogène par les processus services, le service composite doit avoir une description de service. Cet aspect important est mentionné car il est un pivot de la réutilisabilité potentielle d'un composite. Cependant il n'est pas encore traité dans notre approche.

Pour offrir cette meilleure compréhension de la composition, le méta-modèle identifie les dépendances entre ces différents éléments. Il apporte ainsi une perspective nouvelle sur les travaux étudiés en explicitant les conséquences potentielles des uns sur les autres. En spécifiant ces interdépendances, nous proposons une méthode de coordination de tous ces aspects. Cette gestion de la coordination est ce que nous nommons le processus d'*auto-composition*.

L'auto-composition est le processus qui fournit à notre service composite la capacité de modifier sa composition de services en exploitant de façon coordonnée les différents aspects précédents, réifiés au niveau architectural. Ainsi, en fonction des modifications du contexte, le service composite cherche à adapter sa composition en retirant ou ajoutant de nouveaux services (préoccupation (c)). À cause de la variété des implémentations possibles, ces nouveaux services ne sont pas forcément directement exploitables par

le composite. Ils nécessitent potentiellement une gestion particulière afin de pouvoir communiquer avec eux et tirer profit de leurs fonctionnalités (préoccupation (b)). Enfin, le composite gére les conséquences des changements sur la collaboration de la nouvelle composition (préoccupation (a)). L'ensemble devra garantir l'isolation des différents clients qui ne devront pas être affectés par des modifications ne les concernant pas (préoccupation (d)).

Le méta-modèle de service composite vise la qualité de réduction du couplage entre les services constituants sélectionnés. Cet objectif de couplage faible conditionne les éléments qui seront extraits des différentes approches de compositions proposées pour chacun des aspects. De ce fait, les éléments impactant sur d'autres qualités telles que la performance ne sont pas choisis prioritairement.

Le reste du chapitre s'organise de la façon suivante. La section 2.2 se focalise sur les éléments architecturaux du méta-modèle de service composite qui réifient les aspects importants d'une composition de services. La section 2.3 définit les mécanismes et les dépendances entre ces différents éléments architecturaux. Elle décrit le fonctionnement du processus d'auto-composition et son impact sur la réduction du couplage entre services constituants. La section 2.4 illustre le fonctionnement du service composite sur une instanciation simple. Enfin la section 2.5 conclut le chapitre.

2.2 Éléments architecturaux

Notre méta-modèle est abordé comme une extension du méta-modèle de *Service* de l'OASIS [OAS08]. Il se base sur la réification des différents aspects et propriétés d'une composition de services. Chacune de ces propriétés est clairement identifiée et définie. Puis nous les associons à un élément architectural précis de notre méta-modèle.

2.2.1 Vue globale

Ainsi, un service composite est un service. Il en hérite donc l'ensemble des propriétés classiques [OAS08] telles que les notions de description de service ou de fonctionnalités (figure 2.1). Par opposition à un service atomique, un service composite est une composition d'un ou plusieurs services : les *services constituants*. Ce découpage en services constituants et services composites représente le découpage des niveaux de description : constituant ou composite.

Lors de son invocation, le service composite sous-traite son exécution à ses services constituants de façon transparente pour l'utilisateur. Nous répartissons ces services constituants en *services métiers* et *services gestionnaires* du composite :

- *services métiers* : services fonctionnels ou non fonctionnels (sécurité, etc.) qui sont sélectionnés pour répondre aux besoins de l'architecte lors de la définition de son composite. Ces services fournissent leurs fonctionnalités sans connaissance globale sur la composition où ils participent. Les services métiers sont réunis dans le *Service Composite Métier* ou SCM.

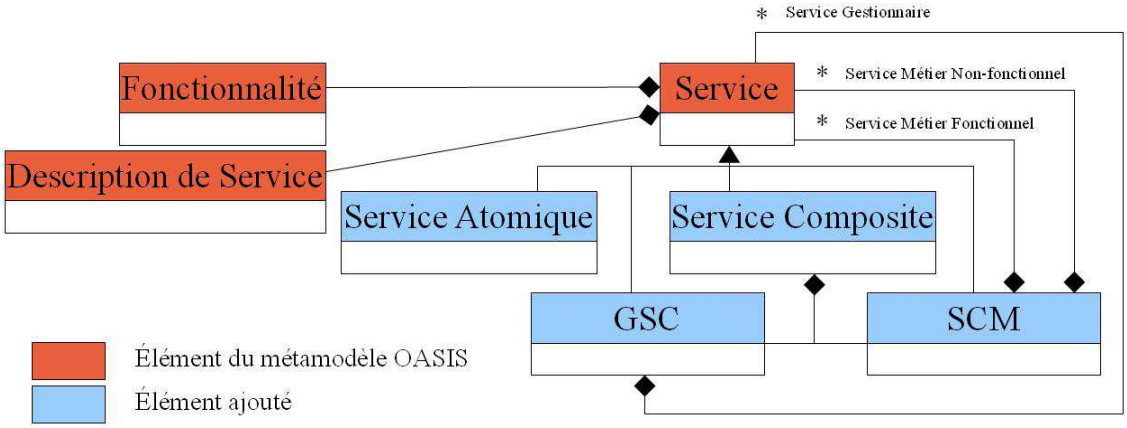

Fig. 2.1 – Méta-modèle de service composite : vue globale

– *services gestionnaires* : services spécialisés dans la gestion des logiques de composition. Ils gèrent les autres services constituants et ont donc une connaissance globale de la composition. Les services gestionnaires sont réunis dans le *Gestionnaire de Service Composite* ou GSC.

Ainsi, un service composite peut être vu comme un ensemble de services métiers et de services gestionnaires. Les services métiers réalisent les tâches définies par l'architecte et les services gestionnaires assurent les bonnes coopérations entre services constituants. Cette distinction est rendue visible au niveau architectural par le service composite qui est constitué des SCM et GSC (cf Figure 2.1). De plus, cette classification des services constituants offre une répartition claire en trois niveaux d'exposition :

– le premier niveau correspond à la partie fonctionnelle qui porte concrètement sur la tâche que doit réaliser le composite. Il regroupe les services métiers fonctionnels dont le résultat de l'exécution est directement visible pour le client du service composite car il représente le produit commercial attendu ;
– le second niveau correspond à la partie non fonctionnelle. Il regroupe les services métiers qui sont chargés de la sécurité, de la confidentialité, de l'authentification des clients du composite, etc. Ce niveau a une importance variable pour le client en fonction de ses besoins et ces différents aspects peuvent être visibles ou non ;
– le dernier niveau correspond à la partie gestionnaire. Il regroupe l'ensemble des services gestionnaires qui assurent la mécanique opératoire entre tous les services constituants. Ces aspects ont peu d'intérêt pour le client et sont en général invisibles.

Notre travail d'explicitation d'une composition de services se focalise principalement sur la définition du GSC. Nous identifions clairement les caractéristiques de gestion attendues et les réifions au niveau de l'architecture de notre service composite en tant que services gestionnaires. Nos contributions portent non seulement sur cette clarification mais aussi sur la définition des interdépendances entre les rôles des gestionnaires. Par la suite, ces interdépendances seront la base de la définition du mécanisme d'auto-composition qui est la coordination de l'ensemble de ces gestionnaires.

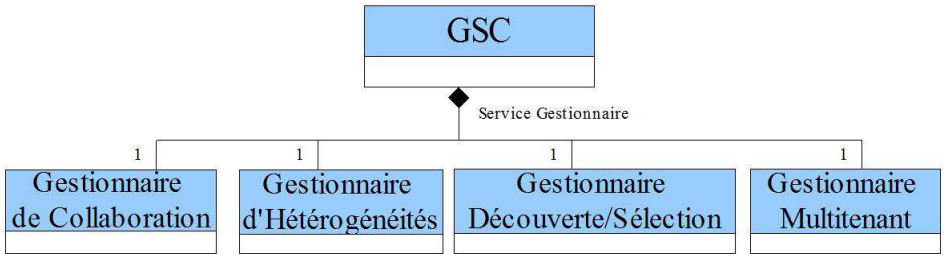

Fig. 2.2 – GSC : gestionnaire de service composite

2.2.2 GSC : Gestionnaire de service composite

Le GSC réunit l'ensemble des services gestionnaires qui sont totalement transparents aux utilisateurs. Il représente la partie invisible du composite. Cette partie est chargée des mécaniques opératoires liées aux logiques de composition, c'est-à-dire la manière dont sont gérés les services métiers. À partir des travaux existants autour de la composition de services, présentés dans le chapitre 1, nous pouvons abstraire quatre rôles principaux :

- la *gestion des collaborations* : la capacité du composite à connaître les services métiers à invoquer et à coordonner ces invocations [PLS08, MRD08] ;
- la *gestion des hétérogénéités* : la capacité du composite à pouvoir communiquer avec les services métiers et de solliciter leurs fonctionnalités de façon correcte. Cette gestion regroupe les préoccupations d'invocation et de médiation. L'invocation est la capacité du composite à déclencher l'exécution des services constituants [RdBM+06, OAS09]. La médiation est la capacité du composite à assurer la bonne compréhension des données échangées entre ses services constituants [RdBM+06, BP08] ;
- la *gestion des contextes et des adaptations* : la capacité du composite à modifier son architecture, c'est à dire ajouter ou retirer des services métiers constituants en fonction du contexte environnemental, qu'il soit système ou utilisateur [OAS08, BGL07, CDA08, NGM+08].
- la *gestion multitenant* : la capacité du composite à gérer de multiples clients en parallèle en garantissant les isolations de contextes, la continuité d'exécution, etc [Jac05].

Pour supporter ces quatre rôles, nous spécifions quatre services gestionnaires inclus dans le GSC (figure 2.2) :

- le *gestionnaire de collaboration* : qui est en charge des exécutions correctes des fonctionnalités métiers du composite ;
- le *gestionnaire d'hétérogénéités* : qui est en charge des invocations des services métiers concrets en respectant leurs contraintes de réalisation technique ;
- le *gestionnaire de découvertes et sélections* : qui est en charge des découvertes et des sélections de services métiers concrets alternatifs en cas de défaillances ; et
- le *gestionnaire multitenant* : qui est en charge de la gestion des différents contextes des consommateurs du service composite.

2.2.3 Gestionnaire de collaboration

Le gestionnaire de collaboration est responsable de l'exécution des différentes fonctionnalités métiers offertes par le service composite. Chacune de ces fonctionnalités est associée à une coordination particulière des services métiers. Il maintient cette coordination à travers la notion de *schéma de collaboration*. Le schéma de collaboration définit les relations entre services en terme de flots de données et de travail [DP06]. Le flot de données représente les différents échanges de données entre les services. Le flot de travail est associé au flot de données et spécifie l'ordonnancement des invocations de services. Il exprime les concepts de séquencement, parallélisme, exclusion, boucle, etc. Le gestionnaire de collaboration est associé à un moteur d'exécution qui interprète le schéma de collaboration afin d'effectuer les bonnes successions d'invocations des différents services métiers en leur fournissant les données nécessaires. Cette approche est caractéristique de ce qui se fait en services web [ACKM03] avec les fichiers BPEL [OAS07] comme schéma de collaboration.

Dans notre objectif de clarification des caractéristiques liées à un service composite, nous définissons un raffinement du principe de schéma de collaboration classique suivant les niveaux d'abstraction, design-time et runtime, en *type de schéma de collaboration* et *instance de schéma de collaboration*.

2.2.3.1 Type et instance de schéma de collaboration

Un type de schéma de collaboration est un élément du design-time qui décrit une coordination entre les services métiers. Il est spécifié par l'architecte au moment de la définition des fonctionnalités métiers de son composite. Le service composite est ensuite déployé et mis à disposition des utilisateurs potentiels. Au runtime, lorsqu'un client invoque une de ces fonctionnalités, le gestionnaire de collaboration instancie le type de schéma de collaboration correspondant à cette fonctionnalité pour produire une instance de schéma de collaboration. C'est cette instance qui est par la suite interprétée par le moteur d'exécution et qui lui indique la séquence des services métiers à invoquer. Chaque client est donc lié à sa propre instance de schéma de collaboration. Cette approche garantit un isolement des exécutions lors de la gestion de multiples invocations parallèles.

Le principe d'*instanciation* est donc le point central de notre gestionnaire de collaboration. Il est le processus qui assure la production d'une instance de schéma de collaboration à partir d'un type de schéma de collaboration. Ces différents schémas organisent les coordinations entre services suivant les deux niveaux d'abstraction. Leur définition repose sur une clarification de la notion de service au design-time et au runtime en termes de *services abstraits* et *services concrets*.

2.2.3.2 Service abstrait et service concret

Le service abstrait est un élément architectural du design-time. Il est la brique de base de construction d'un type de schéma de collaboration. Le service concret est la représentation du service abstrait au runtime. Il représente le service réel, déployé à

distance et invocable, qui est concrètement utilisé par le service composite. Il est la brique de base de construction d'une instance de schéma de collaboration. Lors de son interprétation par le moteur d'exécution, l'instance de schéma de collaboration fournit donc l'ensemble des services concrets qui sont réellement utilisés.

Cependant, la production d'un service concret à partir d'un service abstrait n'est pas assurée par le processus d'instanciation. Le passage du design-time au runtime repose sur les processus de *découvertes* et *sélections dynamiques de services*. Le service abstrait est donc l'expression formalisée des besoins de l'architecte qui définit l'ensemble des caractéristiques qu'il recherche. Ces caractéristiques représentent les bases de réflexion qui ont permis à l'architecte de construire les différents types de schéma de collaboration lors de la spécification des fonctionnalités de son service composite.

Le service abstrait est donc une description abstraite de services. Il pose les critères de recherche qui sont utilisés par les moteurs de découverte et sélection afin d'identifier les services concrets. Ces services concrets sont ensuite utilisés pour instancier les types de schéma de collaboration. Ainsi, le processus d'instanciation des types de schéma de collaboration repose sur les processus de découverte et sélection sur les services abstraits.

2.2.3.3 Vers l'héritage de type de schéma de collaboration

Le découpage en type et instance de schéma de collaboration s'inspire directement du paradigme objet. Ainsi, nous cherchons à profiter des mêmes bénéfices sur la réutilisation. D'une façon similaire à l'héritage de classe, nous allons parler d'*héritage de type de schéma de collaboration*.

L'héritage est particulièrement utile pour le développement de versions spécialisées d'un type de schéma de collaboration et aussi pour assurer la gestion de multiple clients différents. En effet, un des principes fondamentaux des AOS[1] est la possibilité de négociation entre consommateur et fournisseur de services. Outre les négociations sur les éléments non-fonctionnels (qualité de services, sécurité, etc.) un client peut demander un comportement particulier du service composite [ZMCW09, ZM11]. De fait, le type de schéma de collaboration initialement défini par le fournisseur ne correspond pas forcément aux nouveaux besoins. Le processus d'héritage permet de facilement développer des versions spécialisées et d'organiser leurs instanciations. Ainsi, chaque fonctionnalité du service composite est associée à une hiérarchie de plusieurs types de schéma de collaboration en relation d'héritage qui fait le parallèle avec les négociations des différents clients. Lors de l'invocation d'une fonctionnalité par un client, le service composite instancie, dans la hiérarchie d'héritage correspondant à cette fonctionnalité, le type de schéma de collaboration associé à l'identité de ce client.

En résumé, le gestionnaire de collaboration maintient trois informations capitales :

- les différents types de schéma de collaboration qui sont associés aux différentes fonctionnalités métiers attendues du service composite ;
- les hiérarchies d'héritage entre les types de schéma afin de conserver l'organisation des versions spécialisées et leurs bonnes utilisations ;

[1] Architecture-Orientée Services

– et enfin les instances de schéma de collaboration qui représentent les exécutions courantes des types de schéma de collaboration.

Les deux premières catégories représentent des informations persistantes tout au long du cycle de vie du composite. Elles sont nécessaires tant que le service composite est déployé et invocable. La troisième catégorie, qui représente les instances de schéma de collaboration, est non persistante. La création de ces instances est liée aux invocations des clients du composite ; leur maintien par le composite est dicté par leur exécution ; leur destruction est liée à la fin de cette exécution.

Nous représentons chacune de ces informations par des graphes en définissant les types de noeuds et les types d'arcs associés.

– le **graphe de type de schéma de collaboration** : spécifie les flots de travail et de données entre les services métiers abstraits. Ces flots représentent les dépendances fonctionnelles entre ces services. Le flot de données exprime les utilisations d'informations que l'on modélise par le *lien d'utilisation*. Le flot de contrôle représente l'ordre des invocations que l'on modélise par le *lien de précédence*. Ce graphe possède donc les *services métiers abstraits* en type de noeuds, et les *liens de précédence* et *d'utilisation* en type d'arcs (Figure 2.3). Ces différents types d'arcs peuvent être spécialisés pour modéliser les ordonnancements particuliers tels que le parallélisme, les boucles, etc.
– le **graphe de hiérarchie d'héritage et d'instanciation** : spécifie d'une part, les relations d'héritage entre types de schéma de collaboration associés à une même fonctionnalité, et d'autre part, les relations d'instanciation entre les types et les instances de schéma de collaboration. Ce graphe possède donc les *types de schéma de collaboration* et les *instances de schéma de collaboration* en type de noeuds, et les *liens d'héritage* et *d'instanciation* en types d'arcs (Figure 2.3). Le lien d'héritage est entre types de schéma de collaboration. Le lien d'instanciation est entre un type de schéma de collaboration et une instance de schéma de collaboration. Chaque fonctionnalité offerte du service composite est associée à son graphe de hiérarchie d'héritage et d'instanciation.
– le **graphe d'instance de schéma de collaboration** : spécifie les flots de travail et de données entre les services métiers concrets. D'une façon similaire au graphe des types de schéma de collaboration, on modélise ce graphe par les *services métiers concrets* en type de noeuds, et les *liens de précédence* et *d'utilisation* en type d'arcs (Figure 2.3).

Ainsi, le gestionnaire de collaboration est en charge des exécutions des fonctionnalités métiers du service composite. La capacité d'observer ses exécutions est obligatoire pour détecter d'éventuelles défaillances liées aux logiques métiers telles que des résultats incohérents ou des mauvaises successions d'invocations de services. Pour supporter ces détections de violations de contraintes d'exécution, un *gestionnaire d'observation des exécutions* est inclus dans le gestionnaire de collaboration.

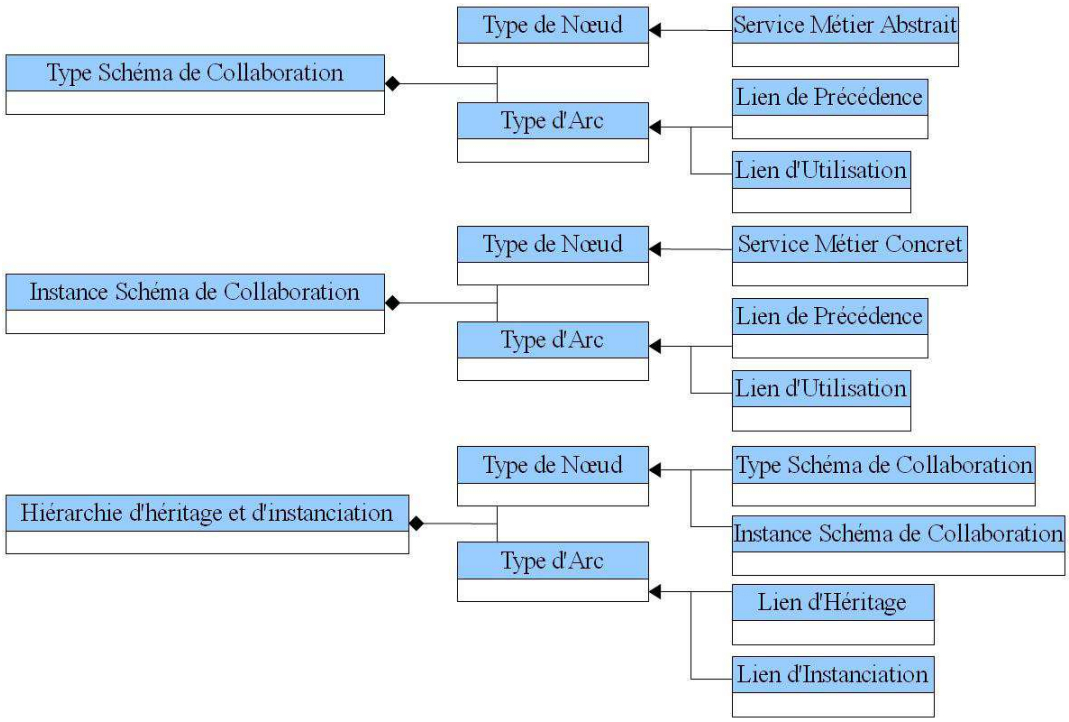

Fig. 2.3 – Gestionnaire de collaboration : graphes

2.2.3.4 Gestionnaire d'observation des exécutions

Le gestionnaire d'observation des exécutions est responsable de la surveillance de l'exécution des différents schémas de collaboration. Ainsi, chaque schéma de collaboration est associé à un moniteur particulier qui collecte et analyse les informations liées à son utilisation. L'ensemble des moniteurs est regroupé dans le gestionnaire d'observation. Pour spécifier ces moniteurs, nous réutilisons les notions de *moniteur de classe* et de *moniteur d'instance* [BGPT09] et les définissons dans notre contexte.

Un **moniteur de classe** observe le comportement des types de schéma de collaboration associés à une fonctionnalité du composite. Il regroupe les informations sur l'historique d'utilisation de ces types de schéma telles que le nombre d'instanciation, les exécutions parallèles, etc. Le moniteur de classe est donc responsable de l'observation d'un graphe de hiérarchie d'héritage et d'instanciation (section 2.2.3.3).

Le **moniteur d'instance** observe l'exécution d'une instance de schéma de collaboration. Il analyse les informations fonctionnelles telles que les pré et post-conditions. Ainsi, il est capable de détecter les possibles dysfonctionnements des services métiers concrets. Le moniteur d'instance est donc responsable de l'observation d'un graphe d'instance de schéma de collaboration (Section 2.2.3.3).

Chaque moniteur a un cycle de vie lié à celui du graphe qu'il observe. Ainsi, les moniteurs de classe sont des éléments persistants tandis que les moniteurs d'instances sont créés puis détruits, en parallèle avec la création et la destruction des instances de

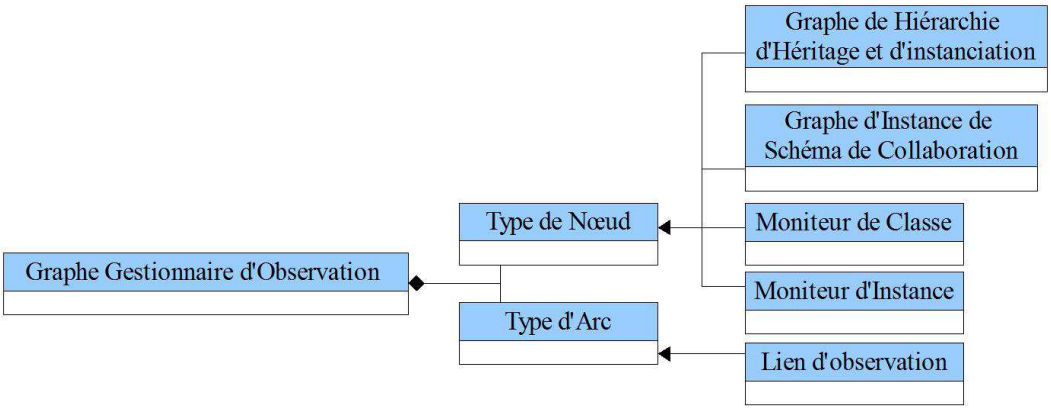

Fig. 2.4 – Gestionnaire d'observation : graphe

schéma de collaboration. Le gestionnaire d'observation des exécutions est donc associé à un générateur de moniteurs. Ce générateur produit des moniteurs d'instances en fonction des différentes créations d'instances de schéma de collaboration.

À son tour, le fonctionnement du gestionnaire d'observation peut être représenté par un graphe qui maintient les associations entre les graphes du gestionnaire de collaboration et les moniteurs de classe ou d'instance. Ce graphe possède donc le *graphe de hiérarchie et d'instanciation*, le *graphe d'instances de schéma de collaboration*, le *moniteur de classe* et le *moniteur d'instance* en type de noeuds, et le *lien d'observation* en type d'arcs. Un moniteur de classe est associé à un graphe de hiérarchie et d'instanciation par un lien d'observation. De la même façon, un moniteur d'instance est associé à un graphe d'instance de schéma de collaboration (Figure 2.4).

2.2.4 Gestionnaire d'hétérogénéités

Le gestionnaire d'hétérogénéités est responsable des communications avec les services métiers concrets durant l'exécution des instances de schéma de collaboration par le gestionnaire de collaboration. Son rôle est de garantir l'invocabilité des services concrets, c'est-à-dire la capacité du service composite de comprendre et d'être compris par ces mêmes services constituants de façon à assurer les bonnes utilisations.

L'ensemble des hétérogénéités peut être divisé en trois catégories (section 1.4.2) :

- les *aspects technologiques* : cette catégorie regroupe les problèmes de compatibilité relatifs aux technologies utilisées telles que les protocoles de transports de messages, les formats et cryptages de données, les technologies d'implémentations, etc. [OAS09, KRB+07]
- les *aspects conversationnels* ou comportement extérieur : cette catégorie regroupe les problèmes de compatibilité relatifs aux comportements extérieurs des services concrets réutilisés. De fait, un service impose une succession précise d'appels de ses fonctions pour l'utiliser de façon correcte. L'établissement du schéma de conversations appropriées entre le service et son client est donc nécessaire.

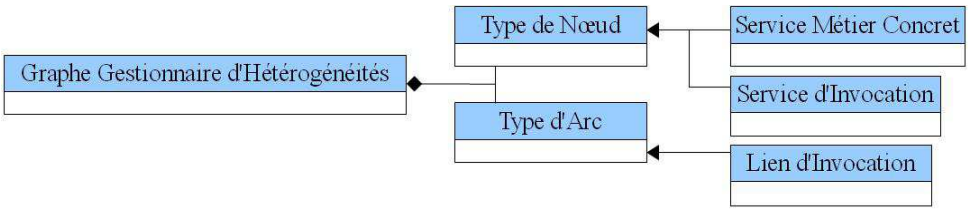

Fig. 2.5 – Gestionnaire d'hétérogénéités : graphe

[CNP09, LFW+08]

– les *aspects données échangées* : cette catégorie se focalise sur la compatibilité des données échangées d'un point de vue sémantique et syntaxique, c'est-à-dire la bonne compréhension du sens attaché aux données. Par exemple, un integer qui représente une date peut être au format anglophone (mois, jour puis année) ou francophone (jour, mois puis année), ou encore une valeur exprimée en euros ou en dollars. [CRZ09, NVSM07]

Le principe du gestionnaire d'hétérogénéités est de combiner certains aspects des travaux précédents sur la gestion des hétérogénéités pour générer dynamiquement des entités logicielles capables de résoudre ces problèmes. Il est donc associé à un générateur de *service d'invocation*. Un service d'invocation est un service qui agit en tant que proxy entre le service composite et un service métier concret. Il est responsable de la communication avec ce service concret et gère les trois aspects de compatibilité précédents. Chaque service métier concret utilisé par le service composite est donc associé à un service d'invocation qui lui est dédié.

D'une façon homogène au travail qui est fait pour le gestionnaire de collaboration, le gestionnaire d'hétérogénéités est modélisé par des graphes. Chaque graphe associe un service concret à son service d'invocation. Ainsi, il possède les *services métiers concrets* et *services d'invocation* comme type de noeuds, et le *lien d'invocation*, d'un service d'invocation vers un service métier concret, comme type d'arcs (Figure 2.5).

Le service d'invocation a un rôle d'intermédiaire, entre le service composite et les services métiers concrets utilisés, qui le place dans une position favorable à l'observation des communications. Il s'assure du respect des contrats qui ont été négociés entre le service composite et les fournisseurs des services métiers. Ainsi, il est responsable des observations unitaires, sans prise en compte de la composition globale, qui concernent les aspects particuliers au service métier cible. Par exemple, il signale tous services non atteignables, c'est-à-dire les services ne répondant pas aux invocations dans un temps imparti. De plus, il s'assure du respect du formatage des données reçues, du cryptage, de l'authenticité, etc.

Ainsi, l'aspect d'identification des services défectueux est supporté par les moniteurs de classes et d'instances et les services d'invocation. Les services d'invocation ont une observation centralisée sur les services métiers pris de façon unitaire, tandis que les moniteurs ont une approche haut niveau sur l'analyse des logiques de collaborations.

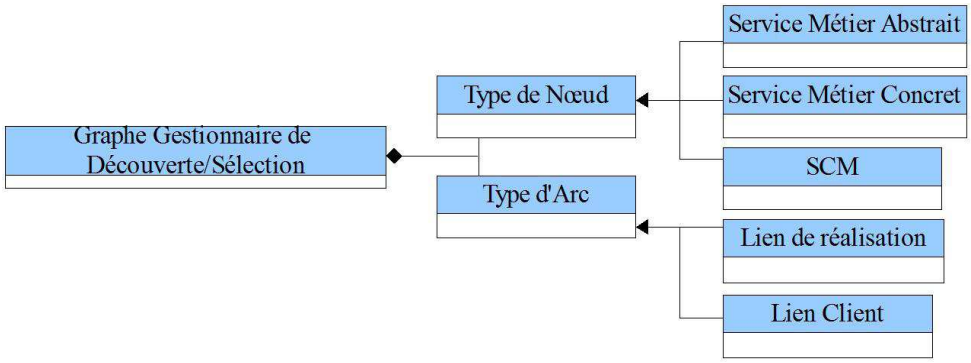

Fig. 2.6 – Gestionnaire de découverte et sélection : graphe

2.2.5 Gestionnaire de découverte et sélection

Le gestionnaire de découverte et sélection est responsable de l'identification de solutions alternatives aux services métiers concrets préalablement identifiés comme défectueux. Un service est dit défectueux s'il ne respecte plus le contrat établi avec ses clients. Le gestionnaire de découverte et sélection exécute à nouveau les processus de découverte et sélection sur le service abstrait associé au service concret défectueux. Il identifie de nouveaux services candidats qui peuvent répondre aux exigences et sélectionne le plus adapté. Le gestionnaire de découverte et sélection est donc associé à un moteur de découverte et sélection de services.

Comme pour les gestionnaires de collaboration et d'hétérogénéités, nous modélisons le fonctionnement du gestionnaire de découverte et sélection par un graphe. Ce graphe maintient la relation entre les services métiers abstraits et les services métiers concrets actuellement utilisés pour les réaliser. Le service composite via son SCM (Service Composite Métier) est donc client de ces services métiers concrets. Ainsi, le graphe du gestionnaire de découverte et sélection possède les *services métiers abstraits*, les *services métiers concrets* et le *SCM* comme type de noeuds, et le *lien de réalisation* et le *lien client* comme type d'arcs. Le lien de réalisation est entre un service métier abstrait et un service métier concret. Le lien client est entre le SCM et les services métiers concrets (Figure 2.6).

Nous favorisons l'utilisation de la notion de lien client plutôt que de lien de composition entre le SCM et les services métiers concrets afin de mettre en perspectives l'absence d'une réelle composition physique au sens classique. En effet, le paradigme service repose sur un principe de responsabilité propriétaire poussé à l'extrême. Le service réutilisé est préalablement déployé et entièrement sous le contrôle de son fournisseur. Son utilisation est dictée par ce fournisseur au travers de sa description de service. Les utilisateurs du service n'interagissent avec lui que via invocation de l'interface contractuelle.

2.2.6 Gestionnaire multitenant

Le gestionnaire multitenant est responsable de la gestion des différents contextes clients. En effet, un service est par définition multitenant, c'est-à-dire qu'il est voué à gérer de multiples clients et connexions en parallèle. Le service composite doit être à son tour capable de gérer plusieurs clients. Le gestionnaire multitenant est donc l'entité logicielle chargée de conserver les informations nécessaires à l'exécution correcte des différentes fonctionnalités. Il sauvegarde les éléments tels que l'identité des clients, les données liées à l'exécution, les historiques d'utilisations, etc. Le gestionnaire multitenant maintient en particulier les informations relatives aux contrats d'utilisation établis après négociation entre le fournisseur du service composite et ses différents clients. Il conserve les relations entre l'identité d'un client et les types de schéma de collaboration qui lui sont associés. En effet, un client peut avoir demandé un comportement fonctionnel spécialisé qui nécessite un type de schéma de collaboration différent. Ces relations sont donc nécessaires au gestionnaire de collaboration pour produire les instances de schéma de collaboration appropriées lors des invocations.

En résumé, le GSC est donc composé de quatre services gestionnaires (Figure 2.7) :
- le *gestionnaire de collaboration* : qui est composé d'un moteur d'exécution d'instances de schéma de collaboration et d'un gestionnaire d'observations des exécutions. Le gestionnaire d'observation est lui-même composé d'un générateur de moniteurs et des moniteurs de classe et d'instance ;
- le *gestionnaire d'hétérogénéités* : qui est composé d'un générateur de service d'invocation et de l'ensemble des services d'invocations utilisés ;
- le *gestionnaire de découverte et sélection* : qui est composé d'un moteur de découverte et sélection de services.
- le *gestionnaire multitenant* : qui est chargé de la gestion des contextes des différents clients ;

La section suivante présente la coordination de l'ensemble des services gestionnaires par le GSC. Nous détaillons les mécaniques opératoires qui permettent : d'une part, le déroulement classique d'un service composite dans un scénario sans aucun service défaillant ; puis d'autre part, le déroulement du processus d'auto-composition pour supporter les adaptations contextuelles dynamiques.

2.3 Mécanismes et dépendances

Le GSC regroupe l'ensemble des services gestionnaires qui sont responsables des logiques de composition. Son rôle est de coordonner ces gestionnaires afin d'assurer l'exécution des fonctionnalités offertes par le service composite. De plus, en cas de défaillances sur les services métiers concrets réutilisés, le GSC doit organiser les actions des différents services gestionnaires de façon à permettre au service composite d'identifier ces services défaillants, de trouver une solution alternative à ces dysfonctionnements, et de mettre en pratique cette solution en conservant la cohérence de départ afin de garantir

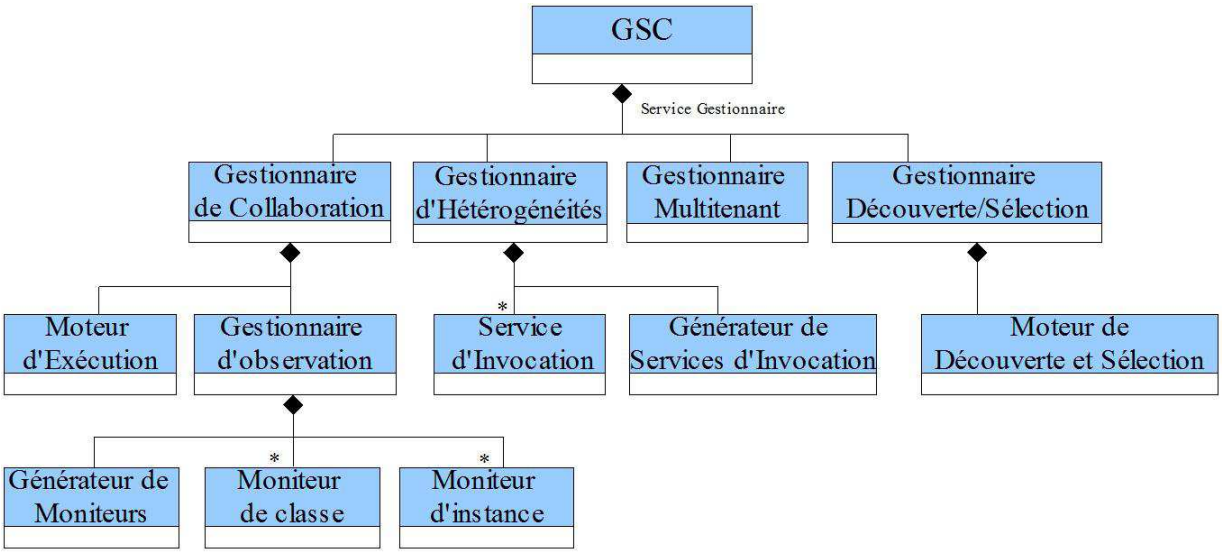

Fig. 2.7 – GSC et services gestionnaires

la continuité d'utilisation.

2.3.1 Mécanique opératoire sans défaillance

Nous supposons que le service composite est préalablement déployé. Les processus de découverte et sélection de services ont déjà identifié les services métiers concrets qui correspondent aux besoins définis par l'architecte. Le gestionnaire d'hétérogénéités a généré les services d'invocation nécessaires à la communication avec ces services concrets à partir de leur description de service.

2.3.1.1 Exécution simple

Lors de l'invocation d'une des fonctionnalités du service composite par un client, le message est transmis au GSC (figure 2.8). Le GSC demande au gestionnaire de collaboration d'instancier le type de schéma de collaboration associé à la fonctionnalité voulue. L'instance de schéma de collaboration est ensuite interprétée par le moteur d'exécution du gestionnaire de collaboration. Au cours de cette exécution, les messages vers l'extérieur passe par le GSC. Ainsi, les demandes d'invocation sur les services métiers concrets sont interceptées par le GSC qui les transmet au gestionnaire d'hétérogénéités. Le gestionnaire d'hétérogénéités utilise le service d'invocation approprié pour communiquer avec le service concret demandé. Ce service invocation supporte les aspects technologiques, conversationnels, et sémantiques des données liés aux exigences du service réutilisé.

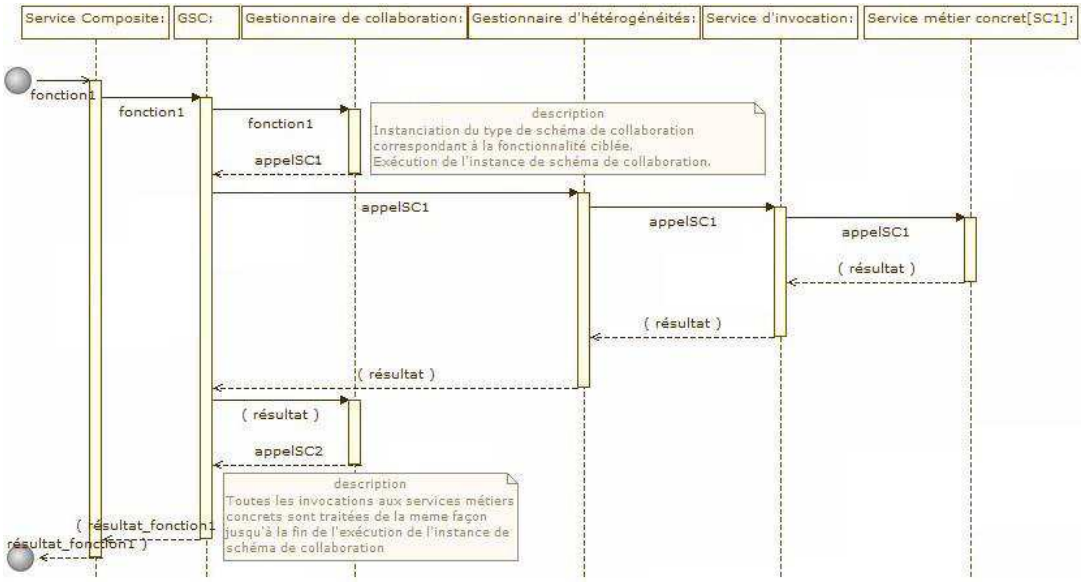

Fig. 2.8 – Exécution simple

2.3.1.2 Exécution multitenant

L'exécution multitenant inclut le gestionnaire multitenant dans la mécanique opératoire décrite précédemment. Lors de l'invocation d'un client sur le service composite, le GSC demande au gestionnaire multitenant de s'assurer de l'identité du client (figure 2.9). Dans le cas d'une identification positive, ce dernier charge le contexte de ce client. Il analyse le contrat établi lors des négociations avec le fournisseur du service composite afin de déterminer le type de schéma de collaboration associé à la fonctionnalité invoquée. En effet, une même fonctionnalité peut être associée à plusieurs types de schéma de collaboration. Le nombre de types de schéma de collaboration dépend du nombre de versions spécialisées du comportement de cette fonctionnalité qui ont été négociées par les différents clients du service composite. Après avoir identifié le type de schéma de collaboration correspondant à la demande, le GSC invoque le gestionnaire de collaboration qui instancie ce schéma et commence l'exécution de l'instance produite. Les demandes d'invocations de services métiers concrets sont ensuite transmises au gestionnaire d'hétérogénéités à travers le GSC. Le gestionnaire d'hétérogénéités est en charge, via les services d'invocations appropriés, des communications réelles avec les services concrets réutilisés.

L'autre rôle du gestionnaire multitenant est de maintenir un certain nombre d'informations, définies par l'architecte, relatives aux exécutions demandées par les clients. Ainsi, en parallèle à l'instanciation d'un type de schéma de collaboration, le gestionnaire de collaboration demande à son gestionnaire d'observation de générer un moniteur d'instance. Le moniteur d'instance observe l'exécution de l'instance de schéma de collaboration pendant que le moniteur de classe, lié à la fonctionnalité ciblée, observe les actions sur

le graphe de hiérarchie d'héritages et d'instanciation. Ces deux moniteurs sont chargés de la récupération des données pertinentes sur l'exécution et des demandes de mises à jour du contexte client faites au gestionnaire multitenant via le GSC. Le moniteur d'instance est uniquement lié à un client particulier. Le moniteur de classe gère les multiples instanciations faites par l'ensemble des clients.

Les mécaniques opératoires précédentes présentent des scénarios d'éxécution sans aucune défaillance sur les services métiers concrets. Pour gérer ces éventuels dysfonctionnements, nous détaillons dans les sections suivantes les dépendances entre les services gestionnaires qui servent de bases au processus d'auto-composition.

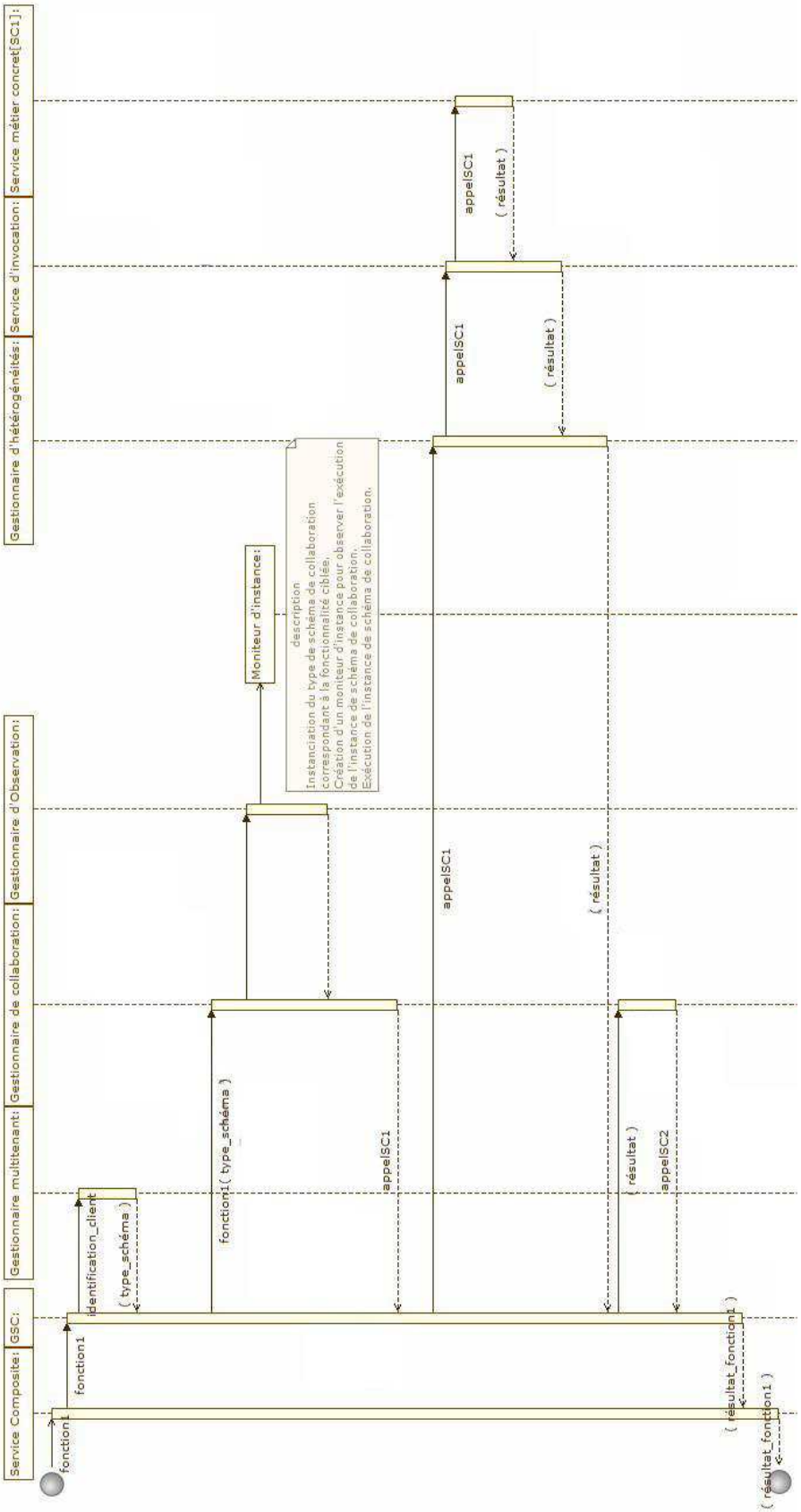

Fig. 2.9 – Exécution multitenant

2.3.2 Dépendances entre gestionnaires

La modélisation des différents graphes maintenus par les gestionnaires de collaboration, d'hétérogénéités et de découverte et sélection est faite de façon homogène. Les noeuds des graphes sont des services constituants présents dans notre méta-modèle. Les arcs représentent des dépendances entre ces noeuds. Cette spécification des types de services gestionnaires et de leurs rôles offre une vision claire du service composite. De plus, elle est la base pour l'explicitation de l'ensemble de leurs dépendances imbriquées. Ces dépendances permettent la définition du mécanisme d'*auto-composition* : la capacité du composite à modifier son architecture en fonction du contexte et d'assurer la consistance avec les logiques de composition.

Ainsi, les modifications (retraits et ajouts) sur l'ensemble des services métiers concrets d'un service composite ont des conséquences directes sur les logiques de composition. Les rôles d'invocation et de collaboration portés par les services gestionnaires dépendent des services métiers présents dans le composite, et cette présence des services métiers est sous l'influence du gestionnaire de découverte et sélection.

De la même façon qu'il est responsable des exécutions sans défaillance, le GSC est naturellement en charge du mécanisme d'auto-composition. Ainsi, après l'identification par les différents moniteurs ou les services d'invocation des services métiers concrets défectueux, le GSC invoque le gestionnaire de découverte et sélection. Ce dernier retire ces services du composite puis utilise son moteur de découverte et sélection pour trouver des remplaçants à partir des services abstraits associés. Ces changements sont ensuite répercutés sur le graphe du gestionnaire de découverte et sélection. Enfin, le GSC doit gérer les impacts possibles de ces changements sur les autres gestionnaires :

- le gestionnaire de collaboration : le nouvel ensemble de services métiers concrets peut nécessiter des flots de travail et de données différents et donc modifier les instances de schéma de collaboration.
- le gestionnaire d'hétérogénéités : le nouvel ensemble de services métiers concrets implique des nouveaux services d'invocation afin d'assurer les communications.

Ainsi, nous abordons la gestion des dépendances suivant ces deux catégories : les dépendances du gestionnaire de découverte et sélection vers le gestionnaire de collaboration, puis du gestionnaire de découverte et sélection vers le gestionnaire d'hétérogénéités.

2.3.2.1 Du gestionnaire de découverte et sélection vers le gestionnaire de collaboration

Durant l'exécution d'une instance de schéma de collaboration, des services métiers concrets peuvent être identifiés comme défectueux. Cette identification est sous la responsabilité de deux types d'éléments architecturaux de notre méta-modèle :

- le moniteur d'instance, associé à l'instance de schéma de collaboration, qui observe le respect de différentes contraintes d'exécution définies par l'architecte ;
- ou les services d'invocation utilisés lors des appels aux services métiers concrets qui s'assurent de la validité de la communication (temps de réponse, formatage des

données, sécurité, etc.)

Lorsqu'un service métier concret lui est signalé comme défectueux, le GSC transmet l'information au gestionnaire de découverte et sélection. Dans un premier temps, le gestionnaire de découverte et sélection retire de son graphe le noeud associé à ce service métier concret. Le lien client vers le SCM et le lien de réalisation vers le service métier abstrait qui lui sont associés sont retirés à leur tour. Le service métier abstrait se trouve donc déconnecté du graphe principal. Ainsi, le gestionnaire de découverte et sélection cherche à rétablir cette connexion. Il utilise son moteur de découverte et sélection sur ce service métier abstrait afin d'identifier de nouveaux services métiers concrets candidats. Il sélectionne le service le plus adapté, parmi l'ensemble des candidats, puis l'ajoute à son graphe en rétablissant les liens client et de réalisation.

Les instances de schéma de collaboration qui utilisent le service métier concret défectueux ne sont plus valides. Elles doivent à leur tour faire le remplacement du noeud concerné par la solution alternative trouvée par le gestionnaire de découverte et sélection. Ainsi, les graphes du gestionnaire de collaboration qui modélisent ces instances doivent retirer le noeud service métier concret défectueux. Cependant, la solution trouvée par le gestionnaire de découverte et sélection peut entraîner des flots de travail et de données différents de ceux définis par le type de schéma de collaboration qui a servi de base à l'instanciation.

En effet, le gestionnaire de découverte et sélection utilise un moteur de découverte et sélection pour identifier les services métiers concrets candidats à la réalisation des services métiers abstraits. Comme présenté dans le chapitre 1, les algorithmes de découverte de services actuels peuvent être classés en deux catégories [KKS07] : les approches 1-1 [GNY04, VGS+05] et les approches 1-N [KKS07, BP08]. Les approches 1-1 identifient un service abstrait à exactement un service concret. Les approches 1-N sont capables de réaliser des compositions de services concrets pour répondre aux besoins exprimés par un service abstrait.

De fait, les approches 1-1 n'ont pas d'impact sur les flots de travail et de données car elles effectuent un remplacement direct du service métier concret défectueux. On peut parler d'une action d'adaptation de *sélection de services* (*service selection* [CCG+08]). Cependant, les approches 1-N supposent que le service métier concret est remplacé par une composition de services. Les flots de travail et de données de l'instance de schéma de collaboration doivent être modifiés afin que le moteur d'exécution du gestionnaire de collaboration soit capable d'interpréter et d'exécuter l'ensemble des invocations nécessaires. On peut parler d'une action d'adaptation de *sélection d'architecture* (*architecture selection* [CCG+08]) où l'on introduit comme solution alternative un ensemble précis d'entités et de relations entre ces entités.

2.3.2.2 Du gestionnaire de découverte et sélection vers le gestionnaire d'hétérogénéités

Le gestionnaire de découverte et sélection définit l'ensemble des services métiers concrets qui sont présents dans le service composite. L'ajout d'un nouveau service

concret doit s'accompagner de la création d'un service d'invocation capable d'assurer les communications avec lui. Dans le cas d'un remplacement de type 1-1, le gestionnaire d'hétérogénéité génère un service d'invocation chargé de la communication avec le nouveau service concret. Dans le cas d'un remplacement du type 1-N, chaque service concret de la composition alternative doit être invocable. N services d'invocation sont donc créés. Ainsi, le gestionnaire d'hétérogénéités crée différents graphes qui représentent les relations d'invocation entre les nouveaux services concrets et les services d'invocation associés.

En résumé, les dépendances entre gestionnaires sont originaires du type de solutions remplaçantes (1-1 ou 1-N) identifiées par le gestionnaire de découverte et sélection.

2.3.3 Le processus d'auto-composition

Le processus d'auto-composition peut se découper en trois étapes :
- l'identification des services métiers concrets défectueux ;
- les découvertes et sélections des services concrets alternatifs ;
- la gestion des impacts de la modification de l'ensemble des services métiers concrets utilisés.

On suppose que le service composite est préalablement déployé. L'ensemble des services métiers concrets nécessaires aux différentes fonctionnalités est déjà établi. Un client invoque une de ses fonctionnalités ce qui entraîne l'instanciation du type de schéma de collaboration approprié. L'instance de schéma de collaboration produite est exécutée dans le gestionnaire de collaboration par le moteur d'exécution. C'est au cours de son exécution que sont détectées des défaillances sur les services métiers concrets.

L'identification des services métiers concrets défectueux est sous la responsabilité de deux entités : les moniteurs d'instance et les services d'invocation.
- moniteur d'instance : chaque instance de schéma de collaboration est associé à un moniteur d'instance. Ce moniteur d'instance observe les flots de travail et de données entre les services métiers concrets. Il s'assure du respect des contraintes d'exécution définies par l'architecte telles que les pré- et post-conditions. Lorsqu'un service concret viole une de ces contraintes, le moniteur d'instance le signale au GSC. Par exemple, dans le cas du non respect d'une post-condition, le moniteur d'instance intercepte cette donnée et transmet au GSC un message de défaillance sur le fonctionnement du service métier concerné. Le GSC prend alors la décision de remplacer ce service.
- service d'invocation : chaque service métier concret est associé à un service d'invocation. Les services d'invocation jouent le rôle de proxy entre le service composite et les services extérieurs et se chargent des communications entre eux. En cas de non respect du contrat sur les communications par le service métier concret, le service d'invocation notifie ce service comme défaillant au GSC. En effet, le service composite est client du service métier concret. Un contrat a donc été préalablement établi avec son fournisseur qui définit la coordination de la communication, le format des données, le type de cryptage, etc. Le service d'invocation s'assure du respect de ce contrat. En cas de notification de violation du contrat par le service d'invocation

au GSC, ce dernier prend alors la décision de remplacer le service métier concret.

Pour effectuer le remplacement d'un service métier concret, le GSC invoque tout d'abord le gestionnaire de collaboration afin de stopper l'exécution de l'instance de schéma de collaboration concernée. L'arrêt d'une instance s'accompagne d'une sauvegarde des états courants nécessaires à une reprise de l'exécution juste avant l'appel au service défectueux. Ces états sont sauvegardés dans le contexte du client maintenu par le gestionnaire de multitenant. L'instance de schéma de collaboration est ensuite détruite, ainsi que le moniteur d'instance associé. En parallèle, le fait qu'une de ses instances a été stoppée représente le type d'information observé par le moniteur de classe.

Le GSC invoque par la suite la fonctionnalité de remplacement du gestionnaire de découverte et sélection en lui fournissant le service métier concret ciblé. Ce dernier utilise son moteur de découvertes et sélections pour identifier la solution alternative, de type 1-1 ou 1-N. Le GSC transmet les descriptions des nouveaux services métiers concrets au gestionnaire d'hétérogénéités qui se charge de la génération des services d'invocation nécessaires. Le GSC demande au gestionnaire de collaboration une nouvelle instanciation du type de schéma de collaboration en lui fournissant les nouveaux services concrets et les possibles conséquences sur les flots de données et de travail d'origine. Cette instanciation s'accompagne classiquement de la création d'un moniteur d'instance. Enfin, le GSC récupère auprès du gestionnaire multitenant les états sauvegardés précédemment. Ces états sont transmis au gestionnaire de collaboration afin de faire redémarrer l'instance de schéma de collaboration à l'étape qui convient.

2.3.4 Vers la réduction du couplage

La définition de notre méta-modèle a pour objectif premier d'expliciter les concepts et mécanismes associés à la composition de services à travers la notion de service composite. Cependant, la manière dont sont modélisés ces concepts et la façon de les organiser les uns par rapport aux autres sont dirigées par une volonté de réduction du couplage. De fait, notre méta-modèle cherche à spécifier un service composite qui réduise au maximum les dépendances entre ses services constituants.

2.3.4.1 Bénéfices du couplage faible

Le principe de couplage faible est un objectif de qualité central des architectures orientées services [OAS08, PH07, Kay03], indissociable de la racine théorique du service en terme de flexibilité et d'agilité de l'application. Comme décrit dans le chapitre 1, le SOSE est issu de besoins spécifiques de logiciels agiles capables de s'adapter rapidement à l'environnement, que ce soit pour répondre aux possibles erreurs systèmes ou encore pour suivre l'évolution des exigences métiers des clients ou du fournisseur. Dans ce contexte, le couplage faible entre les services en collaboration qui construisent ces applications est un enjeu capital.

Le couplage faible correspond à la réduction des dépendances entre éléments en relation. Dans le cas d'une composition de service et de la notion de service composite,

les dépendances sont réparties horizontalement, entre les services constituants, et verticalement, entre les services constituants et le service composite qui exploite leurs fonctionnalités. Limiter les *dépendances horizontales* renforce l'isolement de chacun des services constituants et ainsi restreindre les propagations des erreurs. De plus, cette meilleure isolation simplifie les possibles retraits et ajouts de services constituants en réduisant leurs conséquences sur les autres services réutilisés ce qui diminue la complexité de leurs prises en compte. Limiter les *dépendances verticales* c'est agir sur la capacité du service composite à exploiter de nombreux services concrets différents en s'abstrayant de toutes contraintes de réalisations et en se focalisant uniquement sur ses besoins fonctionnels.

2.3.4.2 Gestion du couplage faible dans le méta-modèle

L'élément central de la réduction du couplage est la présence du gestionnaire d'hétérogénéités et des services d'invocation. Leur principe même est de s'abstraire des contraintes de réalisation technique (technologiques, conversationnelles, de données [OAS09, CNP09, CRZ09]) afin de pouvoir réutiliser tous types de services quelque soit leur implémentation. L'objectif est que le service composite ne soit plus limité par sa réalisation. Cet aspect cible les dépendances verticales au niveau des invocations.

Le processus d'auto-composition est un autre élément qui tend à réduire le couplage. Grâce à lui, le service composite est capable d'exploiter les solutions alternatives en cas de défaillances des services concrets réutilisés. L'objectif est que le service composite ne soit plus dépendant de ces solutions concrètes. Ce principe cible à nouveau les dépendances verticales au niveau de la réalisation des services.

Enfin, le service composite cherche à minimiser les communications horizontales en privilégiant les communications verticales. En effet, la répartition des fonctions de gestion entre les différents services gestionnaires fait une séparation claire des préoccupations afin d'isoler leurs propriétés. Par la suite, la coordination de l'ensemble se fait à un niveau de description supérieur où le GSC est l'entité responsable unique. Cette approche suit le même principe d'orchestration que pour les invocations des services métiers où les communications se font exclusivement verticalement, entre le composite et ses constituants, et non pas horizontalement, entre les services constituants. En résumé, le service composite repose principalement sur les communications verticales, à travers les différents niveaux de composites et constituants, puis, les apports du gestionnaire d'hétérogénéités et du processus d'auto-composition en termes de diminution du couplage ciblant ces mêmes communications verticales.

2.4 Exemple d'instanciation

Afin d'illustrer les différentes mécaniques opératoires, nous faisons une instanciation simple de notre méta-modèle de service composite pour l'exemple présenté dans la section 1.4 sur un scénario classique de domotique.

2.4.1 Projection du service composite

Pour rappel, le scénario de domotique utilisé est une écoute de musique à contrôle vocal. Le principe est le suivant : un habitant de la maison déclenche l'exécution du scénario en prononçant le titre d'une chanson qu'il veut écouter. Le système récupère ce choix à travers un ensemble de microphones disposés dans la maison. Ce choix est transmis au jukebox qui le cherche dans sa liste de titres et le diffuse pour l'utilisateur.

Nous représentons ce système par notre service composite. Le service composite est divisé en SCM et GSC. Le GSC est composé des gestionnaires de collaboration, de découverte et sélection, d'hétérogénéités et de multitenant.

- le **gestionnaire de collaboration** : lors de la modélisation de notre service composite, l'architecte doit spécifier les différents types de schéma de collaboration qui correspondent aux fonctionnalités fournies. Ainsi, il doit définir les services métiers abstraits et les flots de données et de travail qui dirigent leurs interactions. Le scénario d'écoute de musique est modélisé par deux services abstraits en collaboration : le service abstrait microphone qui décrit les besoins en un service capable de récupérer les sons émis par la voix d'un utilisateur, et le service abstrait lecteur/diffuseur de musique qui décrit un service capable de lire une musique et d'en diffuser le son. Le gestionnaire de collaboration est donc associé à un graphe (Figure 2.10, noeud *Mus*) qui représente ce type de schéma de collaboration. Ce graphe est composé de deux noeuds services abstraits (*SA1* et *SA2*) et des flots de données et de travail qui précisent l'utilisation du microphone avant le lecteur et le passage du choix de la chanson de l'un à l'autre. De plus, le gestionnaire de collaboration est associé à un autre graphe composé actuellement d'un unique noeud *Mus*. Ce graphe de hiérarchie d'héritage et d'instanciation (section 2.2.3.3) appelé *H1* illustre qu'aucune instance de ce type n'est en cours d'exécution et que le service composite ne possède pas d'autres versions spécialées de cette fonctionnalité.
  En parallèle, le graphe du gestionnaire d'observation illustre qu'un moniteur de classe (noeud *MC1*) est associé au graphe de hiérarchie d'héritage et d'instanciation précédent (noeud *H1*). Cependant, du fait qu'aucune instance de schéma de collaboration est en exécution, il n'y a donc pas encore de moniteur d'instance.
- le **gestionnaire de découverte et sélection** : le gestionnaire de découverte et sélection est responsable de l'identification des services métiers concrets à partir des services abstraits. Après l'exécution des processus de découverte et sélection de services, il a identifié les services concrets *Micro* et *JukeB* comme les plus adaptés aux besoins. Le gestionnaire de découverte et sélection est donc associé à un graphe qui représente ces découvertes. Ce graphe est composé des deux noeuds services abstraits, microphone (*SA1*) et lecteur/diffuseur (*SA2*), et de deux noeuds services concrets, *Micro* (*SC1*) et *JukeB* (*SC2*), liés par un lien de réalisation. De plus, les liens client entre le SCM et les services concrets représentent la relation entre consommateur et fournisseurs de services (Figure 2.10).
- le **gestionnaire d'hétérogénéités** : à partir des services concrets identifiés par le gestionnaire de découverte et sélection pour l'instanciation des différents types de

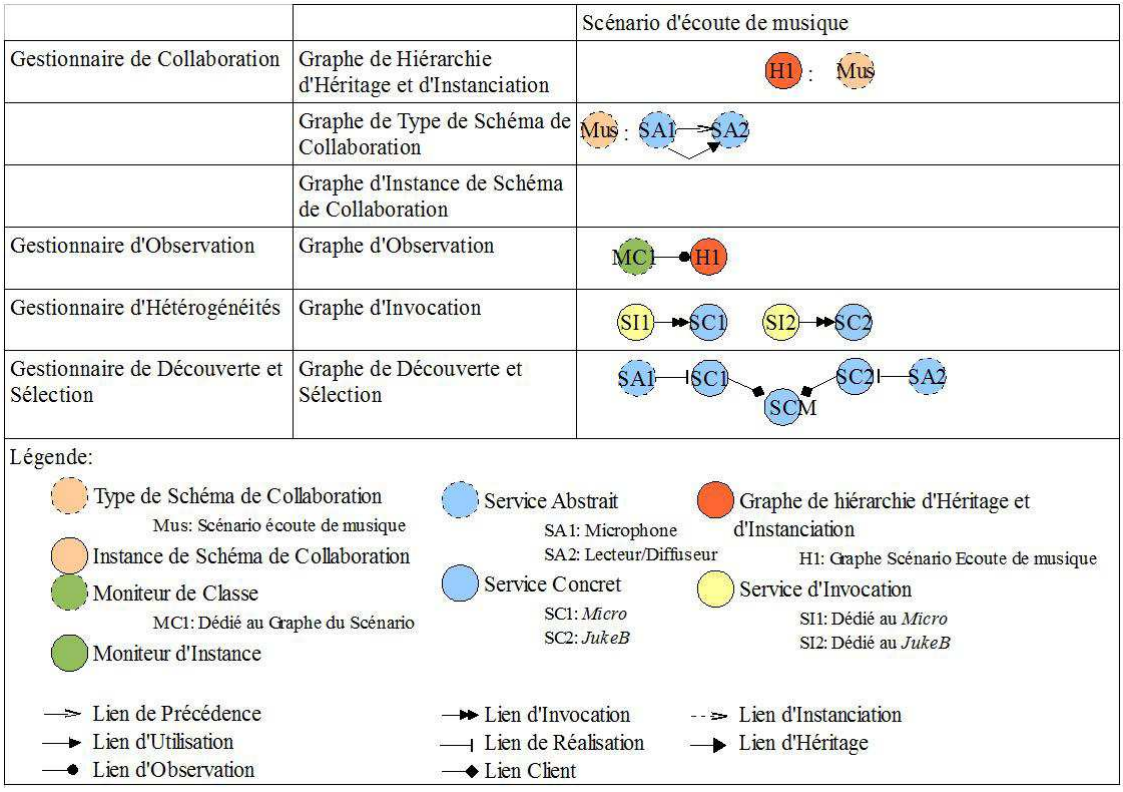

Fig. 2.10 – Scénario domotique : graphes des services gestionnaires

schéma de collaboration, le gestionnaire d'hétérogénéités est chargé de la génération des services d'invocation nécessaires à la communication avec ces services concrets. Ainsi, les services concrets *Micro* (*SC1*) et *JukeB* (*SC2*) sont chacun associés à un service d'invocation (*SI1* et *SI2*) dédié. Le gestionnaire d'hétérogénéités est donc associé à deux graphes. Chaque graphe lie un service concret et son service d'invocation par un lien d'invocation (Figure 2.10, *SI1* avec *SC1* et *SI2* avec *SC2*).

– le **gestionnaire multitenant** : possède l'ensemble des données des différents contextes client. Ces données servent, par exemple, à l'identification des clients, la sauvegarde des éléments d'exécution, la sauvegarde des contrats entre le service composite et ses utilisateurs, etc.

2.4.2 Exécution sans défaillance

Lors de l'invocation de la fonctionnalité d'écoute de musique par un utilisateur, sa demande est transmise du service composite à son GSC qui est chargé de l'exécution. Après s'être assuré de l'identité du client auprès du gestionnaire multitenant, le GSC invoque le gestionnaire de collaboration afin qu'il instancie le type de schéma de collaboration correspondant à ce scénario (c'est-à-dire le noeud *Mus*, figure 2.11). Cette instanciation provoque la création d'un nouveau noeud *Clt1* dans le graphe *H1* de hiérarchie d'héritage

et d'instanciation. Ce noeud représente la nouvelle d'instance du noeud *Mus* qui est le type de schéma de collaboration associé au scénario d'écoute de musique. Cette instanciation correspond à la création d'une instance de schéma de collaboration (*Clt1*) et de son graphe (composé de *SC1* et *SC2*). De plus, un moniteur d'instance (*MI1*) est généré par le gestionnaire d'observation pour assurer l'observation de cette instance (*Clt1*).

Les éléments en rouge de la Figure 2.11 sont ajoutés au cours de cet exemple d'exécution sans défaillance par rapport à l'instanciation de la Figure 2.10.

À présent, nous allons supposer qu'un nouvel utilisateur de ce service composite veut une version personnalisée du scénario d'écoute de musique qui soit capable d'ajouter un jeu de lumière associé aux chansons. La négociation entre ce client et le fournisseur du service composite aboutit à la définition d'un nouveau type de schéma de collaboration (noeud *Lum*, Figure 2.11) qui hérite du précédent (noeud *Mus*). Le graphe *H1* de hiérarchie d'héritage et d'instanciation est modifié en conséquence. Ce nouveau noeud *Lum* s'accompagne d'un nouveau type de schéma de collaboration composé de trois services abstraits *SA1*, *SA2*, et *SA3*. La composition entre *SA1* et *SA2* représente l'élément hérité, l'ajout de *SA3* la spécialisation. Le noeud *SA3* représente le besoin en un service projecteur capable de produire des jeux de lumières en fonction de la musique dont il récupère le flux audio auprès de *SA2*. L'apparition de ce nouveau besoin (*SA3*) déclenche l'invocation du gestionnaire de découverte et sélection afin qu'il identifie un service concret adapté. À la suite du processus de découvertes et sélections, son graphe est modifié par : les ajouts du service abstrait *SA3*, de la solution concrète associée *SC3* (le service concret LCD), d'un lien de réalisation entre eux, et enfin d'un lien client avec le *SCM*. À son tour, le gestionnaire d'hétérogénéité génère un service d'invocation (*SI3*) adapté au service concret LCD (*SC3*) afin d'en assurer les invocations.

À partir de maintenant, lors de l'invocation de la fonctionnalité d'écoute de musique, l'identification du client par le gestionnaire de multitenant s'accompagne de l'identification du type de schéma de collaboration adapté. En effet, si l'invocation est faite par le nouvel utilisateur, le GSC demande au gestionnaire de collaboration l'instanciation du scénario de lumière (*Lum*). Un nouveau noeud *Clt2* (Figure 2.11) représente dans le graphe H1 l'instance de Lum. Cette instance est créée à partir des trois services concrets SC1, SC2 et SC3. En parallèle à cette création, un nouveau moniteur d'instance (*MI2*) lié à *Clt2* est généré afin d'assurer l'observation des exécutions (Figure 2.11).

2.4.3 Auto-composition

Pour illustrer le processus d'auto-composition sur notre exemple, nous allons supposer que le service concret *JukeB* devienne indisponible. Ainsi, lors de l'exécution d'une instance de schéma de collaboration pour le scénario d'écoute de musique, le service d'invocation lié au service concret *JukeB* signale au GSC cette indisponibilité. Le comportement du GSC suit la description faite dans la section 2.3.3. Il demande l'arrêt de l'exécution au gestionnaire de collaboration et invoque la fonctionnalité de remplacement de service concret du gestionnaire de découverte et sélection en lui indiquant le service *JukeB*. Le gestionnaire de découverte et sélection retire le noeud correspondant (*SC2*) de

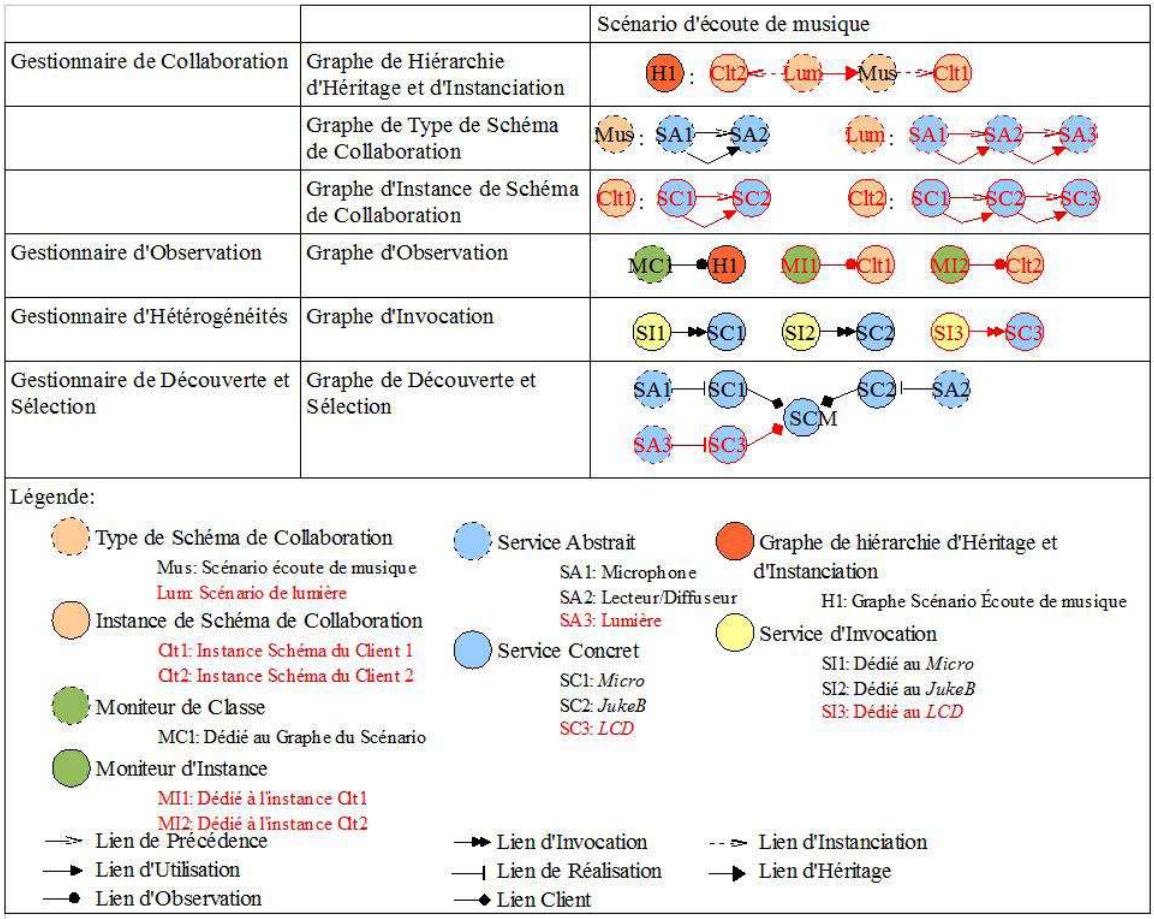

Fig. 2.11 – Héritage et instanciation de type de schéma de collaboration

son graphe et utilise le service abstrait (*SA2*) qui lui était associé comme donnée d'entrée pour son moteur de découverte et sélection. On suppose que l'algorithme de découverte de services utilisé supporte les correspondances de type 1-N. Ainsi, le moteur identifie une composition de deux services capable de répondre aux besoins posés par le service abstrait (*SA2*) : un service de lecteur MP3 (*SC2-1*), pour la sélection et la lecture du titre, et un service d'enceintes (*SC2-2*), pour la diffusion du son.

Par la suite, le fonctionnement de l'auto-composition s'illustre par les modifications des graphes maintenus par les différents services gestionnaires. Le gestionnaire de découverte et sélection modifie son graphe en fonction des résultats de son moteur de découverte et sélection. Il ajoute donc les deux services concrets *MP3* (*SC2-1*) et *Enceintes* (*SC2-2*) puis les associe, d'une part au service abstrait lecteur/diffuseur (*SA2*) par des liens de réalisation, d'autre part au SCM par des liens client (Figure 2.12). Le GSC gère ensuite les impacts de ces modifications sur les graphes des gestionnaires de collaboration et d'hétérogénéités. Le GSC transmet à ces deux services gestionnaires la composition des deux services identifiée.

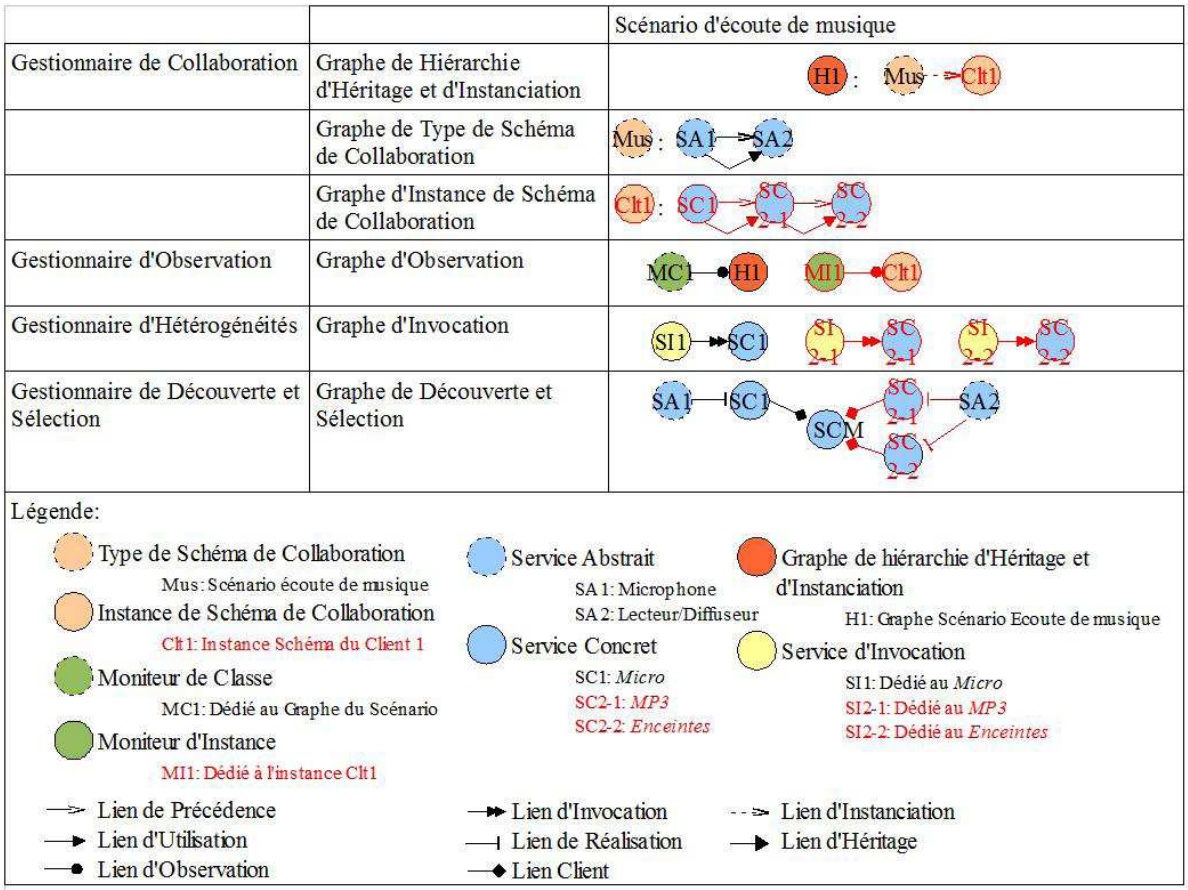

Fig. 2.12 – Exemple d'instanciation : processus d'auto-composition

Le gestionnaire d'hétérogénéité crée deux nouveaux graphes. Un premier graphe où le service concret *SC2-1* est associé par un lien d'invocation avec le service d'invocation *SI2-1* préalablement généré pour assurer les commmunications avec le service composite (Figure 2.12). De même, l'autre graphe associe le service concret *SC2-2* avec son propre service d'invocation *SI2-2*.

Le gestionnaire de collaboration utilise la composition des deux services concrets *MP3* (*SC2-1*) et *Enceintes* (*SC2-2*) pour recréer une nouvelle instance de schéma de collaboration. Le graphe associé qui représente les flots de données et de travail est différent de celui de l'instance d'origine qui avait nécessité le déclenchement de l'auto-composition.

Dans la Figure 2.12, l'ensemble des éléments écrits en rouge représente les modifications par rapport à la Figure 2.10.

2.5 Conclusion

Nous avons présenté dans ce chapitre un méta-modèle de service composite. Ce méta-modèle est à double vocation. D'une part, il propose une meilleure compréhension du

processus de composition de services à travers la spécification d'un service composite. D'autre part, il apporte une vision du service composite qui minimise le couplage entre ses services constituants. L'approche de construction du méta-modèle repose sur une réification au niveau architectural des caractéristiques du processus de composition qui minimisent le couplage entre services en collaboration. Le résultat de cette réification est la spécification d'un ensemble de services gestionnaires dont le rôle est d'assurer les logiques de composition sur les services métiers qui ont été préalablement sélectionnés, puis utilisés pour réaliser les fonctionnalités du service composite. À partir de l'identification de ces services gestionnaires, nous définissons le processus d'auto-composition. Ce processus coordonne les actions des gestionnaires afin d'assurer les adaptations dynamiques du service composite en modifiant son ensemble de services métiers réutilisés.

En amont, la définition du méta-modèle de service composite repose sur les notions de services abstraits et concrets, de types et d'instances de schéma de collaboration et des concepts de moniteur de classe et moniteur d'instance. Ces notions participent à la clarification du SOSE et du processus de composition via cette séparation explicite entre les niveaux d'abstraction, design-time et runtime.

Ainsi, le principe de notre méta-modèle est de réifier les travaux existants et de combiner leurs avantages afin de diminuer le couplage entre les services constituants. Cependant, une mesure claire de cette diminution reste encore à évaluer. En effet, la définition du couplage faible et les moyens de sa mesure sont des domaines relativement peu étudiés dans le paradigme service. De plus, la grande variété des approches de composition qui cherchent à réduire ce couplage pose en conséquence la question de métriques adaptées.

Dans la section suivante, nous proposons une nouvelle définition de la notion de couplage faible. Cette définition pose la base théorique nécessaire à la spécification d'un ensemble de métriques de mesure adaptées au SOSE et à la composition de services. Ainsi, ce travail permet d'évaluer les travaux existants puis de mesurer les bénéfices de notre propre proposition autour du méta-modèle de service composite.

Chapitre 3

Vers une nouvelle définition du couplage faible

3.1 Introduction

Les chapitres précédents ont montré l'importance du processus de composition de services comme socle majeur de la réutilisation. De nombreuses méthodes de composition existent et chacune d'elles propose différents mécanismes pour répondre aux enjeux de dynamicité, d'hétérogénéités des ressources, d'adaptation aux contextes, etc. Afin d'évaluer l'impact de ces propositions, la question sous-jacente "*qu'est-ce qui fait une bonne composition de services ?*" devient critique.

Un des concepts portés par le SOSE qui cherche à répondre à cette préoccupation est un principe de couplage faible entre services en collaboration [Kay03,OAS08,PH07]. Diminuer le couplage c'est réduire les dépendances entre ces services et donc améliorer la flexibilité de la composition globale. Un certain nombre de bénéfices sont immédiatement perçus tels que la localisation des fautes, la facilité de remplacement des services, etc. Cependant cette compréhension intuitive représente quasiment la seule définition du couplage proposée. De plus, les limites des métriques associées tendent à accentuer cette imprécision.

Ainsi, nous cherchons à combler ce manque en proposant une nouvelle définition du couplage faible dans une composition de services. Cette définition a pour objectif de poser les bases pour l'élaboration de métriques qui seront combinées dans une formule d'évaluation du couplage global inspirée des travaux de l'Analyse préliminaire de risques

[Mor02]. La finalité est d'être capable de pondérer de manière objective le couplage d'une composition de services particulière. En mesurant ces compositions spécifiques, notre formule d'évaluation permet leurs comparaisons. Ainsi, elle offre aux architectes une assistance sur le choix de la meilleure composition pour réaliser son système suivant ce critère de couplage faible.

En plus de cette formule d'évaluation pour une composition particulière, nous proposons un cadre de comparaison directement entre méthodes de composition. Ainsi, ce cadre offre des guides supplémentaires sur le choix de l'approche la plus adaptée aux besoins des architectes parmi la multitude proposée. Ce travail sert aussi à l'évaluation de notre méta-modèle où nous pourrons mesurer explicitement ses bénéfices sur le couplage et le comparer à l'existant.

La section suivante motive cette étude par rapport aux définitions existantes du couplage et aux métriques d'évaluation associées. Puis, elle liste les différents objectifs à atteindre. La section 3.3 pose les fondements théoriques de notre vision du couplage et en définit les nouvelles métriques. La section 3.4 présente la formule d'évaluation globale basée sur ces métriques. Le cadre de comparaisons de méthodes de composition est défini dans la section 3.5. Enfin la section 3.6 conclut le chapitre.

3.2 Motivations et objectifs

Les spécifications formelles existantes du paradigme AOS (listées dans [ES08]) telles que [OAS08, Ope09, The08] présentent la notion de couplage faible comme essentielle à une composition de services. La réduction des dépendances entre les services constituants assure des bénéfices immédiats sur la flexibilité et la robustesse d'une application. Cette compréhension intuitive des apports du couplage faible représente l'unique définition apportée par ces spécifications. Ainsi, elles laissent la place à une imprécision récurrente sur les concepts associés au couplage.

3.2.1 Origine des confusions

Au manque de précision des spécifications formelles s'ajoute une confusion dans la façon d'aborder le principe de couplage. Cette confusion de définition est due à la grande variété des approches de composition de services qui se focalisent sur sa réduction. En effet, chacune d'elles fournit de nouveaux mécanismes qui ciblent des aspects particuliers influençant le couplage de la composition résultat. Dans les chapitres 1 et 2 sur l'état de l'art et la définition du méta-modèle, nous avons identifié une liste des caractéristiques d'une composition de services. Cette étude préalable nous permet de classer les approches de composition suivant différents axes principaux de recherche (section 1.4). Parmi ces axes, deux agissent directement sur la réduction du couplage : la *gestion des hétérogénéités* et la *gestion des adaptations contextuelles*.

Les travaux sur la gestion des hétérogénéités se focalisent sur la réduction de la distance entre produits qui semblent différents à première vue. L'objectif est de permettre leur

compatibilité de communication. Dans le cas du SOSE, les produits sont les services, les descriptions de services, les interfaces, etc. Comme expliqué dans les sections 1.4.2 et 2.2.4, les hétérogénéités qui existent entre ces produits peuvent être regroupées en trois aspects : technologiques [OAS09], conversationnels [RdBM+06, KKS07, BP08] et de données [VGS+05, RKL+05].

Ainsi, la recherche sur la gestion des hétérogénéités est extrêmement diversifiée et elle influence les processus fondamentaux du SOSE que sont la publication de services, les découvertes et sélections, et la composition de services.

De son côté, la gestion des adaptations contextuelles en SOSE est la capacité d'une application à effectuer des modifications dynamiques de sa composition. Ces modifications sont des ajouts ou des retraits des services appropriés en réponse à une situation donnée ou par son anticipation. Les travaux sur cette capacité d'adaptation tels que [BGL07, CDA08, CCG+08] cherchent à réduire la dépendance d'une application envers les services qu'elle réutilise en lui permettant l'emploi de solutions alternatives.

Tous ces modèles de composition de services ont des bénéfices intuitivement tangibles sur le couplage. Cependant, les métriques existantes ne permettent pas leurs évaluations précises. Les architectes n'ont pas les moyens de mesurer leurs impacts respectifs. Il est donc difficile d'obtenir un comparatif clair afin de sélectionner l'approche la plus adaptée aux besoins.

Cet état des lieux fut constaté lors de la définition de notre méta-modèle de service composite. En effet, le critère principal qui a dirigé son élaboration fut la réduction maximale du couplage en combinant les bénéfices des approches proposées par la littérature. Cependant, les métriques existantes pour le couplage ne nous permettent pas d'établir de façon claire les contributions du méta-modèle et de le positionner par rapport aux autres propositions.

3.2.2 Imprécisions des métriques existantes

Ces imprécisions ont sans aucun doute favorisé l'apparition de métriques incomplètes d'évaluation du couplage. De fait, les travaux proposés dans la littérature sont incapables de comparer de façon pertinente la diversité des approches de composition.

Dans [PRFT07, GS08], les auteurs réutilisent des métriques établies qui sont issues d'une conception orienté objets [CK94, BWDP98]. Ces métriques ont prouvé leur efficacité pour évaluer ces structures OO, cependant leur parti pris centré sur les concepts liés à l'implémentation par classes n'est pas adapté pour le SOSE [PRF05]. Elles ne peuvent pas mesurer les contributions des travaux présentés précédemment (chapitre 1) sur la gestion des hétérogénéités ou sur la capacité d'adaptation.

Des méthodes alternatives existent [EKM07, MZZW09] mais elles restent incomplètes. En effet, elles reposent sur une analyse des échanges de messages entre services et utilisent des critères tels que le nombre de messages ou encore la complexité des données qu'ils transportent. Elles se focalisent donc uniquement sur l'aspect physique d'une composition de services instanciée et en exécution. À leur tour, elles ne prennent pas en compte les mécanismes liés à la gestion des hétérogénéités et à l'auto-adaptation.

Des travaux similaires sur l'étude du couplage sont menés dans le CBSE tels que [YCR09, GS07]. Cependant, ils ne peuvent être utilisés directement de par les différences théoriques qui existent entre l'orienté composant et l'orienté service (approche exclusivement boite noire des services supportée par les notions de description de service, de fournisseur et de consommateur, etc.), et la présence de processus spécifiques ayant un impact important sur le couplage tel que les découvertes dynamiques de services.

Ainsi, bien que valides, les métriques existantes sont incomplètes et doivent être étendues pour prendre en compte toute la variété des compositions de services proposée par la communauté SOSE.

3.2.3 Expliciter le couplage

Notre objectif est de proposer une méthode d'évaluation du couplage qui prenne en compte les aspects soulignés par les recherches existantes sur la composition de services. Cette nouvelle définition est complétée par des métriques appropriées qui permettent d'en évaluer les concepts. Enfin, ces métriques sont combinées dans une formule d'évaluation globale qui calcule le couplage d'une composition de services particulière.

L'autre aspect de ce travail, en plus d'une nouvelle définition du couplage faible, est de fournir un cadre de comparaison entre approches de compositions de services. Les métriques proposées servent de base à l'identification des différents critères de ce cadre de comparaison. L'objectif final est de fournir une hiérarchie d'approches de composition suivant leur potentiel à réduire le couplage ainsi qu'une évaluation quantitative objective de ce couplage pour les instances composites résultats. Ce travail assiste les architectes dans leur choix de modélisation et de réalisation de leurs applications construites par composition de services.

3.3 La notion de couplage faible

La définition du méta-modèle de service composite repose sur une étude approfondie du mécanisme de composition de services et sur un découpage clair en niveaux d'abstraction : design-time et run-time. Elle s'organise autour des notions de services abstraits et services concrets, type et instance de schéma de collaboration, et type et instance de service composite (section 2.2.3) qui constituent la base de spécification de notre vision du couplage faible.

3.3.1 Localisation des dépendances

3.3.1.1 Vers trois couplages distincts

Le processus de développement d'un service composite peut être divisé en trois phases réparties entre les niveaux d'abstraction :

– la *modélisation* : l'architecte définit le type de service composite. Il exprime ses besoins en terme de services à réutiliser à travers les services abstraits. Puis, il

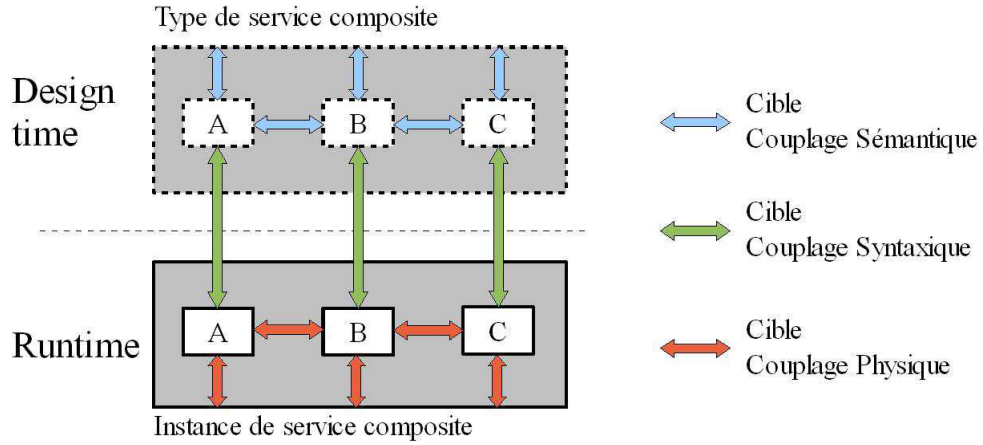

Fig. 3.1 – Cibles des couplages sémantique, syntaxique et physique

modélise les types de schéma de collaboration qui définissent la coordination de ces services abstraits. Cette phase illustre le niveau *design-time*.

- la *composition* : les informations précédentes sont fournies à un moteur de composition de services qui se charge de l'exécution des trois étapes classiques de découverte, sélection puis composition de services (section 1.3.2). Ce moteur réalise l'instanciation du type de service composite. Cette instanciation repose principalement sur l'identification des services concrets qui réaliseront les services abstraits. Cette phase fait le lien entre le *design-time* et le *runtime*.
- l'*exécution* : le composite est déployé puis exécuté. Cette phase illustre le niveau *run-time*.

Sur la base de ces trois phases, nous définissons trois couplages distincts :

- le *couplage sémantique* : ce couplage se base sur la description haut niveau d'un composite qui est défini par l'architecte. Il représente une expertise personnelle sur le domaine d'application particulier du composite. Le couplage sémantique mesure les dépendances entre les services abstraits et leur implication dans les fonctionnalités haut-niveau définies par l'architecte (Figure 3.1) ;
- le *couplage syntaxique* : ce couplage mesure les dépendances en terme de réalisation entre les services abstraits et les services concrets (Figure 3.1) ;
- le *couplage physique* : ce couplage mesure les dépendances entre les services concrets réellement utilisés, en collaboration et en exécution. Il correspond à l'approche classique des travaux existants [PRFT07, GS08, EKM07, MZZW09] (Figure 3.1).

Ce découpage en trois parties distinctes peut s'apparenter à l'approche proposée par [CND90]. Dans ce travail, les auteurs introduisent le paradigme S-F ("*S-F paradigm*" : structure et fonction) qui étend les concepts de l'orienté objets pour améliorer la représentation du design d'un système. Le paradigme S-F traite les structures et les fonctions comme des objets de première classe en modélisant les configurations physiques comme des objets structurels, et le comportement comme des objets fonctionnels. Les auteurs définissent ensuite la notion d'objet interaction qui est chargé de faire le lien

entre les objets structurels et fonctionnels.

Ainsi, notre découpage sémantique, syntaxique et physique s'associe respectivement au découpage fonctionnel, interactionnel et structurel. De fait, le couplage sémantique se focalise sur les services abstraits qui expriment les fonctions recherchées. Le couplage physique se concentre sur les services concrets qui représentent les entités logicielles qui réalisent effectivement la composition. Enfin, le couplage syntaxique étudie le lien entre les services abstraits et les services concrets.

Le choix du vocabulaire sémantique, syntaxique et physique nous a paru plus judicieux que les notions structurelles, fonctionnelles et d'interaction car ces dernières reflètent une vision trop orientée objet pouvant porter à confusion.

3.3.1.2 Exemple

Afin d'illustrer notre découpage, nous utilisons un exemple simple qui se base sur la modélisation du processus de production d'un constructeur de voitures. Dans un premier temps, le constructeur modélise son nouveau prototype. Il décrit les différents composants qui y sont associés, leurs dépendances et la façon dont ils fonctionnent ensemble. Pour la suite du chapitre, nous nous basons sur un ensemble de cinq fonctionnalités classiques attendues d'une voiture.

La *fonction de déplacement* représente la capacité de la voiture à se déplacer. Elle est réalisée par trois composants : le système de refroidissement, le moteur et les roues. Le système de refroidissement assure le bon fonctionnement du moteur qui transmet le mouvement aux roues. La *fonction d'arrêt* représente la capacité de la voiture à stopper son déplacement. Elle est réalisée par deux composants : les freins et les roues. Les freins appliquent une friction sur les roues pour provoquer l'arrêt du mouvement. La *fonction de direction* représente la capacité de la voiture à orienter son déplacement. Elle est réalisée par deux composants : le volant et les roues. Le volant applique une certaine direction aux roues. La *fonction GPS* représente la capacité de la voiture à identifier sa position et à calculer des itinéraires. Elle est réalisée par un seul composant : un GPS. La *fonction air-conditionné* représente la capacité de la voiture à refroidir sa cabine passager. Elle est réalisée par un seul composant : le système de refroidissement de la cabine.

À partir de cette modélisation des fonctionnalités du prototype, le constructeur cherche à sous-traiter la production des différents éléments qui interviennent dans leur réalisation. Ces éléments sont ensuite assemblés pour construire la voiture finale.

Ce scénario classique est facilement modélisable sur une architecture orientée services. L'architecte logiciel doit alors travailler de pair avec les acteurs experts du domaine de l'automobile pour exprimer les besoins du constructeur. Les différents composants de la voiture représentent les services abstraits. La façon dont ces composants s'organisent représente les schémas de collaboration associés aux fonctionnalités qui sont offertes par la composition. Par la suite, le constructeur cherche les entreprises (les fournisseurs de services) capables de produire ces différents composants. Ces productions sont les services concrets et l'assemblage finale représente le service composite.

La figure 3.2 modélise notre exemple. La voiture est représentée par un service

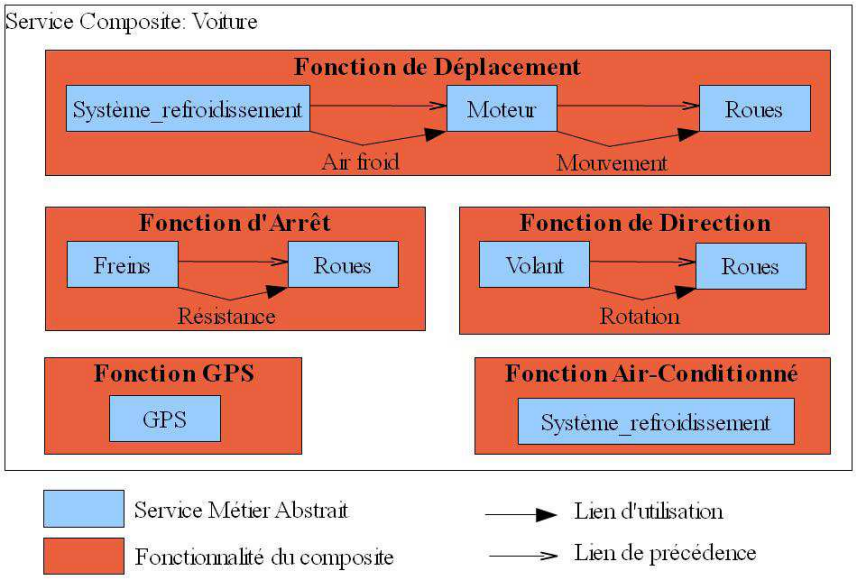

Fig. 3.2 – Modélisation du service composite Voiture

composite qui possède les cinq fonctionnalités requises par l'architecte : *déplacement*, *arrêt*, *direction*, *GPS* et *air-conditionné*. Chaque fonctionnalité est associée à une coordination particulière de services abstraits. Ces services abstraits représentent les différents éléments de la voiture dont la production est sous-traitée. Pour représenter les flots de données et de travail entre ces services abstraits nous réutilisons les notions de type de schéma de collaboration, de lien de précédence et de lien d'utilisation (section 2.2.3). Par exemple, la fonctionnalité de déplacement se base sur une collaboration de trois services métiers abstraits : *Système_refroidissement*, *Moteur* et *Roues*. Le *Système_refroidissement* produit de l'air froid qui est fourni au *Moteur* pour produire du mouvements sur les *Roues*.

Ainsi, en effectuant cette modélisation, le constructeur de voitures spécifie les contraintes sur les services qu'il attend des entreprises sous-traitantes. Chacun des services abstraits et les types de schéma de collaboration associés définissent les caractéristiques recherchées.

3.3.2 Le couplage sémantique

Le couplage sémantique veut rendre explicite l'expertise de l'architecte logiciel sur le domaine d'application de son composite. De manière générale, cette expertise lui est fournie par un acteur de ce domaine. Dans notre exemple, la modélisation d'une voiture passe obligatoirement par une étroite collaboration avec des ingénieurs dédiés à l'automobile.

Le couplage sémantique est localisé au niveau de la phase de modélisation, c'est-à-dire sur la représentation abstraite du composite. Sa mesure est indépendante de

l'implémentation et se focalise sur les fonctionnalités décrites par l'architecte. Le principe du couplage sémantique est d'évaluer l'importance des services abstraits par rapport aux fonctions attendues du composite.

Nous définissons trois niveaux de couplage sémantique :

- le **couplage fort** : un service abstrait et un composite sont fortement couplés au niveau sémantique si ce service participe à une fonctionnalité essentielle du composite. Une fonctionnalité est identifiée comme essentielle de façon subjective par l'architecte. En fonction de son expertise, il définit une fonctionnalité comme majeure c'est-à-dire que sans elle le composite est inutilisable.
  Dans notre exemple, le constructeur de voitures peut définir la fonction de déplacement comme essentielle. Sans ce déplacement une voiture perd de sa substance. Les services abstraits en collaboration (système de refroidissement, moteur et roues) qui supportent cette fonction sont donc fortement liés au composite. Ils représentent des éléments critiques pour le constructeur.
- le **couplage faible** : un service abstrait et un composite sont faiblement couplés si ce service participe à une fonctionnalité non essentielle du composite. Cependant, bien que non essentiels, ces types de fonctionnalités ont un impact direct sur l'efficacité du composite. La qualité globale n'est plus garantie si une ou plusieurs de ces fonctions sont retirées. Nous posons que si toutes ces fonctions non essentielles disparaissent le composite devient inutilisable.
  Dans notre exemple, le constructeur peut définir les services abstraits des fonctions d'arrêt et de direction comme en couplage faible avec le composite. En effet, la voiture devient hors de contrôle si elle perd ces deux fonctions bien que l'essence même d'une voiture qu'est le déplacement soit toujours accessible.
- le **couplage non prédominant** : un service abstrait et un composite sont en couplage non prédominant si ce service participe à une fonction non essentielle du composite et si le retrait de cette fonction n'a aucun impact significatif sur la qualité globale. Toutes les fonctions non prédominantes peuvent être retirées sans porter atteinte aux fonctions essentielles du composite. Elles représentent des caractéristiques optionnelles.
  Dans notre exemple, le constructeur peut définir les services abstraits impliqués dans les fonctions GPS et air-conditionné comme non prédominants car leur dysfonctionnement n'induit pas celui des fonctions essentielles du prototype.

La figure 3.3 représente le couplage sémantique lié à notre exemple et les fonctions du prototype. Chaque service abstrait est associé à sa fonction par les points blancs (couplage non prédominant), noirs (couplage fort) ou gris (couplage faible). Le lien entre les points représente les dépendances horizontales (c'est-à-dire entre les services abstraits constituants) définies par le type de schéma de collaboration en termes de liens de précédence et d'utilisation. Ainsi, l'expertise de l'architecte est clairement exprimée. Les services critiques liés à des fonctionnalités essentielles sont visibles et ceux intervenant dans plusieurs d'entre elles sont facilement identifiables, par exemple les roues ou le système de refroidissement.

Le couplage sémantique représente une identification subjective des services abstraits

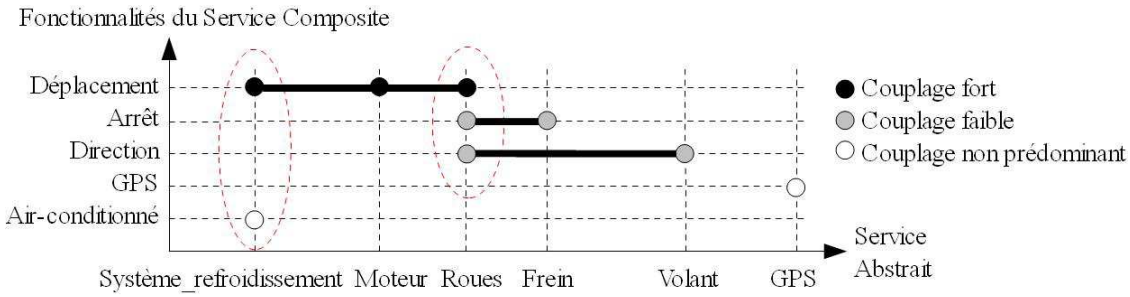

Fig. 3.3 – Couplage sémantique

critiques pour l'architecte.

3.3.3 Le couplage syntaxique

Le couplage syntaxique met en avant une propriété fondamentale du paradigme service qui est la découverte dynamique des services disponibles. Ainsi, ce couplage se concentre sur les dépendances entre les services abstraits, qui sont la base des recherches, et les services concrets, qui remplissent les contraintes recherchées. Comme expliqué dans la section 1.3.2.1, les algorithmes existants de découvertes de services peuvent être classés suivant le type de correspondances qu'ils sont capables d'identifier [KKS07] : 1-1 [GNY04, VGS+05] ou 1-N [KKS07,BP08]. L'approche 1-1 identifie un service abstrait à exactement un service concret. L'approche 1-N identifie un service abstrait à une composition de N services concrets en collaboration.

Nous définissons deux niveaux de couplage syntaxique :

- le **couplage fort** : un service abstrait est en couplage syntaxique fort avec sa solution concrète si cette solution (un service concret ou une composition de services concrets) représente l'unique possibilité de réalisation. Il n'y a pas d'autres solutions alternatives. Ce manque d'alternatives peut avoir deux origines : soit il n'existe pas d'autres solutions qui remplissent les besoins de l'architecte, soit il y a une obligation d'utiliser un service concret particulier.
  De plus, un service abstrait est en couplage syntaxique fort si le service composite ne possède pas de mécanisme d'auto-adaptation. En effet, sans ce mécanisme, les modifications dynamiques de la composition de services sont impossibles. Le système est incapable d'exploiter les solutions alternatives en cas de défaillance d'un des services concrets d'origine, ou si des solutions plus adaptées deviennent disponibles (meilleure qualité de service, faible coût, etc).
- le **couplage faible** : un service abstrait et un service concret sont en couplage faible si il existe plusieurs alternatives concrètes à la réalisation de ce service abstrait et que la composition est capable d'exploiter ces alternatives par un mécanisme d'auto-adaptation. Plus il y a de solutions disponibles, plus le couplage est faible. Un composite qui remplace dynamiquement un service défectueux par une correspondance 1-1 effectue une action d'adaptation de sélection de service

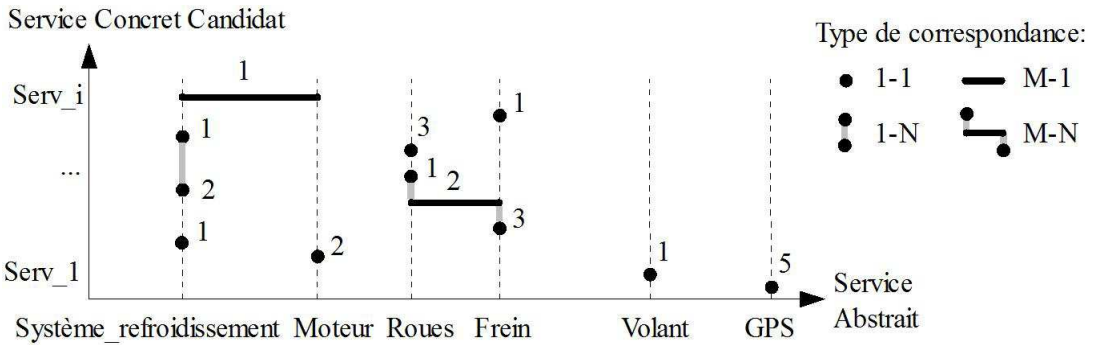

Fig. 3.4 – Couplage syntaxique

[CCG+08]. Un remplacement de type 1-N est une action de sélection d'architecture [CCG+08].

Dans notre exemple, le moteur a un couplage syntaxique fort si une seule entreprise est capable de le produire. Une autre possibilité est que le constructeur de voitures est un contrat d'exclusivité avec une entreprise qui l'oblige à utiliser ses moteurs. Au contraire, il est en couplage faible si de nombreuses entreprises ayant la compétence requise sont connues par le constructeur et que ce dernier soit capable de changer facilement de fournisseurs dans un coût acceptable.

La figure 3.4 représente le couplage syntaxique mesuré à partir des services concrets identifiés par un moteur de découverte particulier. Ces services concrets candidats représentent tous les services qui peuvent être utilisés pour réaliser les différents services abstraits du composite. Un point unique représente une correspondance 1-1 [GNY04, VGS+05] entre un service abstrait et service un concret. Le poids associé est le nombre de services concrets équivalents qui peuvent être utilisés. Il représente le nombre de solutions alternatives qui réalisent la correspondance. Par exemple, le service abstrait GPS peut être réalisé par exactement un service concret et cinq alternatives sont disponibles, c'est-à-dire que cinq services concrets différents sont disponibles pour assurer la correspondance 1-1. De la même façon, une des solutions de réalisation du service abstrait système de refroidissement est une correspondance 1-1 (le point noir isolé) avec une seule possibilité.

Les points liés par une ligne grise verticale représentent une correspondance de type 1-N [KKS07, BP08]. Par exemple, une autre solution de réalisation du système de refroidissement est réalisée par une composition de deux services concrets avec une alternative pour l'un et deux alternatives pour l'autre. De plus cette représentation du couplage syntaxique permet de gérer les correspondances de types M-1 et M-N. Par exemple, les services abstraits système de refroidissement et moteur peuvent être réalisés ensemble par un unique service concret (on peut imaginer un moteur intégrant un mécanisme de refroidissement). Ce service concret particulier est représenté par une ligne noire à cheval sur les deux services abstraits. Le poids associé indique qu'une seule alternative est disponible. Une correspondance de type M-N est défini sur les services

abstraits roues et freins qui peuvent être réalisés ensemble par une composition de trois services, avec l'un d'entre eux à cheval sur les deux abstraits. Dans cette correspondance 2-3 (figure 3.4), la composition des trois services concrets qui réalisent les deux abstraits possède une alternative pour le premier, deux pour le second et trois pour le dernier.

À notre connaissance, il n'existe pas de propositions significatives pour un algorithme de découverte de type M-N. Son étude reste donc un challenge ouvert du SOSE [HkO10b].

Ainsi, le couplage syntaxique exprime les alternatives concrètes et complète le couplage sémantique. En effet, le couplage sémantique permet l'identification des aspects critiques du composite d'un point de vue fonctionnel. Un architecte essaiera de réduire le couplage syntaxique des services abstraits qui ont été identifiés comme critiques à un niveau sémantique. Par exemple, le moteur est défini comme essentiel dans une voiture par le constructeur. Cependant, on peut réduire cette criticité si de nombreuses entreprises sont capables de le produire. En cas de problèmes liés à ce moteur (défauts fabrication, temps de production, etc.) le constructeur peut faire appel à d'autres entreprises.

Cette approche pose donc qu'un service abstrait a un couplage syntaxique d'autant plus faible qu'il possède de nombreuses solutions de réalisation alternatives exploitables. Cette métrique est simplement établie en exploitant les résultat des algorithmes de découvertes utilisés par une composition particulière et dans un contexte particulier. Elle est représentée dans la figure 3.4 par le poids associé à chaque point.

3.3.4 Le couplage physique

Le couplage physique se focalise sur l'implémentation concrète du composite. Cette implémentation correspond à une instance particulière du composite où un choix a été fait sur les services concrets à utiliser. Une solution unique a été choisie pour remplir chacun des besoins exprimés par les services abstraits. Le couplage physique réutilise les recherches existantes et se base sur des mesures empiriques [PRFT07, GS08, EKM07, MZZW09] telles que les appels de méthodes, les nombres de messages échangés, la complexité des messages, le nombre de services liés, etc. Ces métriques permettent l'identification des dépendances sur deux niveaux :

- les *communications horizontales*, directement entre services concrets constituants ; et
- les *communications verticales*, entre le service composite et ses services concrets.

La figure 3.5 représente le couplage physique du composite. Il évalue les dépendances des communications horizontales et verticales entre le service composite et les services concrets qui ont été sélectionnés. Ces services concrets représentent les solutions choisies parmi l'ensemble des possibilités exprimées dans le couplage syntaxique. Pour chaque service de l'axe des ordonnées, nous déterminons les services avec lesquels il communique par un point. Chaque point est associé à un poids calculé en réutilisant les métriques existantes [PRFT07, GS08, EKM07, MZZW09].

Cette représentation graphique identifie facilement les services critiques qui centralisent les communications. Plus le nombre de points sur une ligne est important, plus le service associé à cette ligne est enchevêtré dans la composition. La défaillance de ce

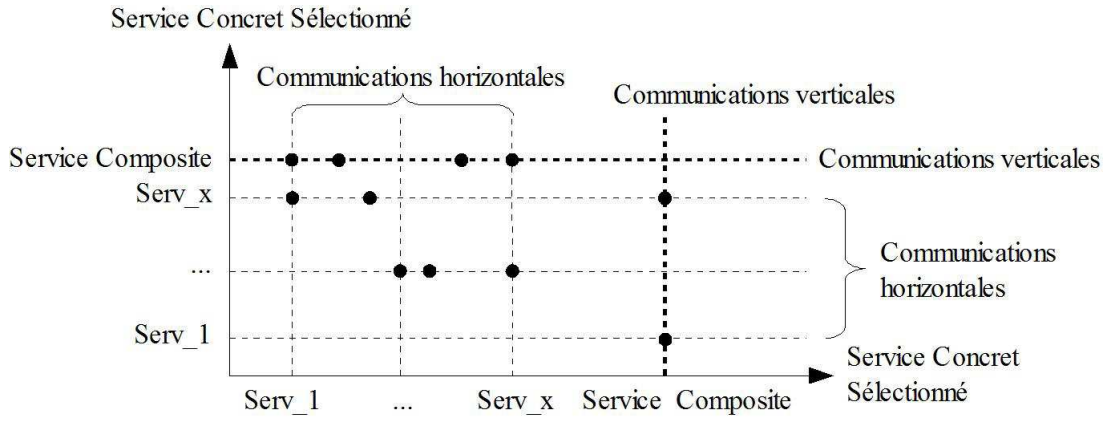

Fig. 3.5 – Couplage physique

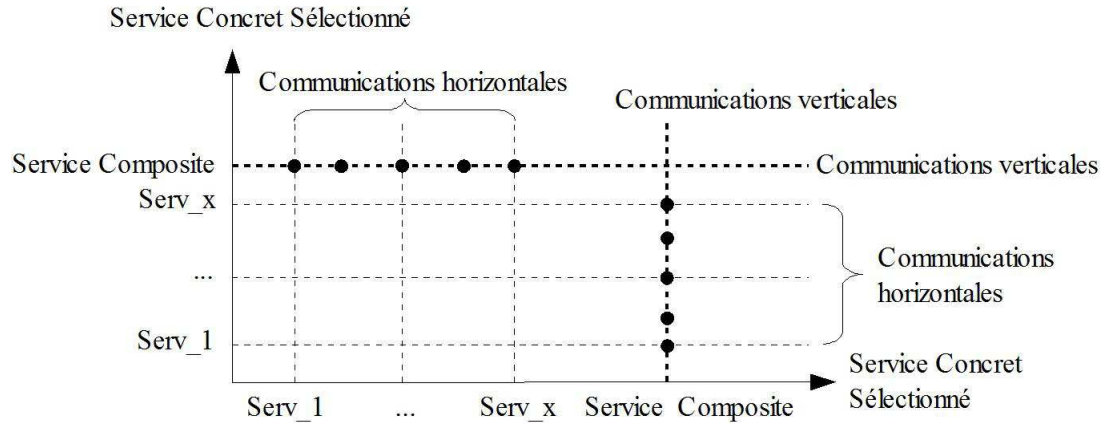

Fig. 3.6 – Couplage physique d'une orchestration

service a donc de nombreuses conséquences sur le fonctionnement des autres intervenants et son éventuel remplacement devient plus complexe à gérer. De plus, ce graphe permet d'appréhender immédiatement la catégorie du type de schéma de collaboration utilisé, orchestration ou chorégraphie. Dans le cas d'une orchestration, l'ensemble des points sont concentrés sur les axes d'abscisse et d'ordonnée liés au service composite, aucune communication directe entre services concrets constituants n'apparaît (Figure 3.6).

Ainsi, nos trois couplages se focalisent sur des aspects différents de la composition de services et agissent de façon complémentaire.

Le couplage sémantique exprime l'expertise de l'architecte et permet l'identification des aspects fonctionnels critiques du composite. Le couplage sémantique agit comme un guide pour le couplage syntaxique. L'architecte cherche à réduire le couplage syntaxique des services abstraits qui sont fortement couplés au niveau sémantique et ainsi rendre cette criticité plus sûre. En effet, plus un service abstrait possède d'alternatives concrètes, moins les risques d'un arrêt total du composite dû à l'absence d'une solution concrète sont importants. Le couplage physique exprime quant à lui les dépendances entre les

services concrets effectivement utilisés par le composite. Il identifie les services concrets problématiques critiques en terme de communications et souligne la complexité de leur remplacement. Le couplage physique peut servir à sélectionner les meilleures compositions concrètes en fonction des besoins.

Cette définition du couplage en trois aspects est la base de notre formule d'évaluation globale du couplage d'un composite particulier.

3.4 Formule d'évaluation globale du couplage d'un composite

Notre formule s'inspire d'une formule existante du domaine de l'Analyse préliminaire de risques [Mor02] qui est couramment utilisée dans l'industrie automobile. Cette formule simplifiée permet de mesurer le risque de défaillance d'un composant d'une voiture.

$$Risk = A * B * C \tag{3.1}$$

A est la *Criticité* d'un composant de la voiture, B est la *Probabilité d'occurence* d'une défaillance sur ce composant et C est la *Probabilité de non-détection* de cette défaillance. À partir de là, nous associons cette notion de risque à notre vision du couplage. En effet, l'essence du couplage est l'expression des dépendances qui peuvent exister entre deux éléments et le principe de dépendance définit que l'un ne peut fonctionner sans l'autre. Réduire le risque que le rôle défini par un service abstrait ne puisse plus être assuré c'est diminuer la dépendance de l'application par rapport à ce service abstrait et donc c'est réduire son couplage. Le calcul de ce risque prend en compte l'ensemble des caractéristiques qui influencent le couplage en redéfinissant les trois variables A, B et C en fonction des couplages sémantique, syntaxique et physique.

3.4.1 Criticité : A

Dans l'automobile, la criticité d'un élément est évaluée par une échelle empirique basée sur l'expérience du constructeur. Cette expérience lui permet d'en exprimer la dangerosité. Cette vision correspond directement à notre approche du *couplage sémantique* qui représente l'expertise de l'architecte. Ainsi, l'élément évalué est le service abstrait, et l'échelle empirique utilisée est en trois valeurs : couplage fort (F), faible (f) et non prédominant (np).

Comme présenté dans la section 3.3.2, l'attribution du couplage sémantique à un service abstrait s'effectue sur la base de l'importance accordée par l'architecte suivant les différentes fonctionnalités auxquelles ce service participe. Par exemple, il peut participer à deux fonctionnalités différentes et avoir un couplage fort pour l'une et un couplage faible pour l'autre. Nous définissons donc une formule qui soit capable de combiner les différentes valeurs de couplage sémantique pour un même service abstrait.

Dans un premier temps, l'architecte doit choisir une borne Δ, entier positif, qui représente la valeur maximale possible pour le couplage sémantique d'un service abstrait.

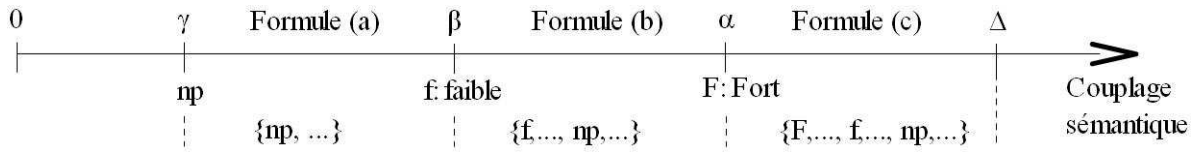

Fig. 3.7 – Répartition des résultats des formules de criticité

Puis, il attribue un poids aux différents couplages, respectivement α, β, γ pour les couplages sémantiques fort, faible et non prédominant, compris entre les bornes 0 et Δ avec $\Delta > \alpha > \beta > \gamma \geq 0$. Ainsi, un service abstrait intervenant dans une seule fonctionnalité en couplage fort aura une valeur égale à α. Si cet unique couplage est faible il sera égale à β ou à γ si il est non prédominant.

Notre formule de combinaison des différents couplages fort, faible et non prédominant est ensuite définie afin de calculer des valeurs résultats en trois groupes distincts :

- **(a)** si le service abstrait est associé uniquement à des couplages non prédominants, la valeur résultat sera comprise entre $[\gamma, \beta]$ (figure 3.7).
- **(b)** si le service abstrait est associé à des couplages non prédominants et faibles, la valeur résultat sera comprise entre $[\beta, \alpha]$ (figure 3.7).
- **(c)** si le service abstrait est associé à des couplages non prédominants, faibles et forts, la valeur résultat sera comprise entre $[\alpha, \Delta]$ (figure 3.7).

Ce principe de répartition des résultats est en accord avec les définitions des couplages sémantiques fort, faible et non prédominant de la section 3.3.2. En effet, quelque soit le nombre de services abstraits non prédominants, associés à des fonctionnalités non prédominantes, que l'on retire de la composition il n'y aura pas d'impact sur la qualité globale, alors que le retrait d'un unique service abstrait en couplage faible impacte sur cette qualité. De la même façon, le retrait d'un unique service abstrait en couplage fort arrête automatiquement le fonctionnement de la composition et donc son impact reste supérieur à la combinaison des retraits de tous les services en couplage faible.

Ainsi, nous définissons un ensemble de trois formules (3.2), (3.3) et (3.4) pour respectivement les cas (a), (b) et (c).

$$\textbf{(a)} : \{np_1, ..., np_n\} = \gamma + \frac{\beta - \gamma}{n} * (n-1) \tag{3.2}$$

$$\begin{aligned}\textbf{(b)} : \{f_1, ..., f_m, np_1, ..., np_n\} = \beta + \frac{\alpha - \beta}{m} * (m-1) \\ + \frac{(\frac{\alpha-\beta}{m+1} * m) - (\frac{\alpha-\beta}{m} * (m-1))}{n+1} * n\end{aligned} \tag{3.3}$$

$$\begin{aligned}\textbf{(c)} : \{F_1, ..., F_r, f_1, ..., f_m, np_1, ..., np_n\} = \alpha + \frac{\Delta - \alpha}{r} * (r-1) \\ + \frac{(\frac{\Delta-\alpha}{r+1} * r) - (\frac{\Delta-\alpha}{r} * (r-1))}{m+1} * m \\ + \frac{\frac{(\frac{\Delta-\alpha}{r+1} * r) - (\frac{\Delta-\alpha}{r} * (r-1))}{m+2} * (m+1)}{n+1} * n \\ - \frac{\frac{(\frac{\Delta-\alpha}{r+1} * r) - (\frac{\Delta-\alpha}{r} * (r-1))}{m+1} * m}{n+1} * n\end{aligned} \tag{3.4}$$

Le tableau 3.1 illustre un certain nombre de résultats possibles en posant $\Delta = 4$, $\alpha = 3$, $\beta = 2$ et $\gamma = 1$:

- **(a)** colonne 1 : représente des combinaisons de valeurs toujours égales à np (couplage sémantique non prédominant). Quelque soit le nombre de np, la valeur résultat sera toujours inférieure à la borne $\beta = 2$ qui est la valeur de f (couplage sémantique faible).
- **(b)** colonne 2 : représente des combinaisons de valeurs égales à np ou f. Quelque soit le nombre de f et de np, la valeur résultat sera toujours inférieure à la borne $\alpha = 3$ qui est la valeur de F (couplage sémantique Fort). De plus, comme pour le cas (a), f sera toujours dominant quelque soit le nombre de np. Par exemple $\{f, np, ..., np\} < \{f, f\}$.
- **(c)** colonne 3 : représente des combinaisons de valeurs égales à np, f ou F, avec F qui sera toujours dominant quelque soit le nombre f et de np. Enfin, quelque soit le nombre de F, f et np, la valeur résultat sera toujours inférieure à la borne $\Delta = 4$.

La criticité A d'un service abstrait cible est donc :

$$A \in \{(a), (b), (c)\} \tag{3.5}$$

Ainsi, on permet le calcul de la criticité d'un service abstrait ciblé, suivant l'expertise de l'architecte et dans le contexte particulier de son composite par rapport aux différentes fonctionnalités dans lesquelles ce service abstrait intervient.

3.4.2 Probabilité d'occurrence d'une défaillance : B

La probabilité d'occurrence d'une défaillance calcule la probabilité que les besoins exprimés par un service abstrait cible ne soient plus assurés, c'est-à-dire qu'il n'existe plus aucune solution concrète capable de remplir les exigences voulues.

Tab. 3.1 – Exemple de calcul de la criticité

(a) : $[\gamma = 1, \beta = 2]$	**(b)** : $[\beta = 2, \alpha = 3]$	**(c)** : $[\alpha = 3, \Delta = 4]$
$\{np\} = \gamma = 1$	$\{f\} = \beta = 2$	$\{F\} = \alpha = 3$
$\{np, np\} = 1,5$	$\{f, np\} = 2,25$	$\{F, np\} = 3,125$
$\{np, np, np\} = 1,666$	$\{f, np, np\} = 2,333$	**{F,np,...,np} < {F,f} = 3,25**
$\{np, np, np, np\} = 1,75$	**{f,np,...,np} <{f,f} = 2,5**	$\{F, f, np\} = 3,291$
$\{np, np, np, np, np\} = 1,80$	$\{f, f, np\} = 2,583$	$\{F, f, np, np\} = 3,305$
{np, ..., np} < β = 2 = {f}	**{f,f,np,...,np} < {f,f,f} = 2,666**	**{F,f,np,...,np} < {F,f,f} = 3,333**
	$\{f, f, f, f\} = 2,75$	$\{F, f, f, f\} = 3,375$
	{f,...,f} < α = 3 = {F}	**{F,f,...,f} < {F,F} = 3,5**
		$\{F, F, F\} = 3,666$
		{F,...,F} < Δ = 4

Cette probabilité B utilise les informations des couplages syntaxique et physique. Le couplage syntaxique calcule le nombre potentiel des solutions alternatives pour la réalisation d'un service abstrait. De fait, les besoins exprimés par un service abstrait ne sont plus remplis si toutes ses alternatives sont indisponibles. De plus, ce poids est pondéré par le couplage physique C_{phys} de la solution concrète actuellement utilisée par la composition. En effet, le couplage physique mesure les dépendances en termes de communications entre les services concrets utilisés. Plus un service est utilisé, plus grande est sa probabilité d'avoir une défaillance. La probabilité B est donc égale à :

$$B = P_{allfail} * C_{phys} \tag{3.6}$$

$P_{allfail}$ est la probabilité que toutes les solutions concrètes d'un service abstrait soient défaillantes. Cette probabilité est fortement liée au nombre de ces solutions. Ce nombre de solutions dépend lui même du moteur de découverte utilisé, donc du type de correspondances abstrait-concret pris en charge pour ce moteur, et de la gestion des hétérogénéités du système (section 3.3.3).

Nous posons P_s la probabilité de défaillance d'un service concret. Nous posons α le poids identifié dans le couplage syntaxique (Figure 3.4) qui représente le nombre d'alternatives de services concrets à un service abstrait. Nous décomposons la formule d'évaluation de $P_{allfail}$ en fonction des types correspondance 1-1, 1-N ou M-N. Bien qu'aucun travail existant ne propose d'approche M-N, ce type de correspondance représente le cas général.

1-1 : un service abstrait est réalisé par exactement un service concret. Ce service concret possède α alternatives (figure 3.4, le service abstrait GPS possède une correspondance 1-1 avec un α égal à cinq). $P_{allfail}$ est la probabilité que toutes ces alternatives soient défaillantes, d'où :

$$P_{allfail} = (P_s)^{\alpha}$$

1-N : un service abstrait est réalisé par une composition de N services concrets au maximum. Les combinaisons possibles varient de 1-1, 1-2 à 1-N. Chacune de ces combinaisons peut avoir un certain nombre d'alternatives représentées par des α sur chacun de ses services concrets impliqués. On obtient donc la formule suivante :

$$P_{allfail} = \prod_{i=1}^{N} \sum_{j=1}^{i} ((P_s)_{ij})^{\alpha_{ij}}$$

M-N : bien que ne possédant pas encore de solution réelle, la correspondance M-N représente une des évolutions naturelles des thématiques de la découverte de services. Les combinaisons de correspondances possibles (1-1, 1-2, ..., 4-6, ... M-N) dépendent des valeurs de M et N, mais aussi du nombre X de services abstraits recherchés par l'architecte dans son composite (figure 3.3, l'architecte est à la recherche de six services abstraits pour modéliser sa voiture). Nous posons λ comme le nombre de ces combinaisons possibles de M dans X. Le calcule de ce λ nécessite donc la prise en compte de deux cas : si $M < X$ et si $M >= X$. Par la suite λ est utilisé pour calculer le nombre de combinaisons de réalisations possibles des services abstraits. On obtient la formule générale suivante :

$$P_{allfail} = \prod_{k=1}^{\lambda} \prod_{i=1}^{N} \sum_{j=1}^{i} ((P_s)_{kij})^{\alpha_{kij}} \tag{3.7}$$

$$M < X : \qquad \lambda = \sum_{i=X-M}^{X-1} \mathrm{C}_{X-1}^{i}$$

$$M \geq X : \qquad \lambda = 1 + \sum_{i=1}^{X-1} \mathrm{C}_{X-1}^{i}$$

Dans notre formule générale, si M est fixé à 1 on retrouve la formule identifiée de 1-N. Si à son tour, dans la formule 1-N, N est égale à 1 alors on retrouve la formule associée à 1-1. Ces trois formules respectent donc le principe d'inclusion entre 1-1, 1-N et M-N.

$P_{allfail}$ est ensuite pondérée par le couplage physique C_{phys}. Le taux d'utilisation d'un service concret et les possibles propagations d'erreurs sont liés aux informations telles que le nombre d'appels, les états communs, les échanges de messages. Plus le couplage physique est fort, plus le coefficient C_{phys} est important.

La formule globale de la probabilité d'occurrence de défaillance B est donc :

$$B = \left(\left(\prod_{k=1}^{\lambda} \prod_{i=1}^{N} \sum_{j=1}^{i} ((P_s)_{kij})^{\alpha_{kij}} \right) * C_{phys} \right) \tag{3.8}$$

Cette formule calcule la probabilité que les fonctionnalités recherchées, exprimées par un service abstrait, ne soient plus assurées, c'est-à-dire que l'ensemble des solutions concrètes disponibles soient défaillantes. Le nombre de ces solutions concrètes varie en fonction du contexte système courant qui regroupe : le moteur de découverte utilisé, les contraintes fonctionnelles et non fonctionnelles posées par le service abstrait et le système utilisé, et enfin la disponibilité actuelle des services concrets.

La prise en compte de ces solutions alternatives suppose donc que le service composite soit capable de modifier dynamiquement son architecture et donc d'en faire un bon usage. Dans le cas contraire, cette probabilité prend en compte uniquement l'instance choisie et pas ces alternatives. La formule de la probabilité d'occurrence de défaillance B est réduite à :

$$B = P_s * C_{phys} \tag{3.9}$$

Dans ce cas particulier, P_s représente la probabilité de défaillance de la solution concrète courante, utilisée par le service composite pour réaliser le service abstrait cible.

Après avoir redéfini les opérandes A et B de la formule globale de l'analyse préliminaire de risques, la section suivante spécifie le troisième et dernier élément, la probabilité C de non détection d'une défaillance.

3.4.3 Probabilité de non détection d'une défaillance : C

La probabilité de non détection d'une défaillance se base sur l'étude du système d'observation du contexte mis en place par un service composite auto-adaptable (c'est-à-dire un composite capable de détecter des services concrets défaillants et de les remplacer par les alternatives). Cette évaluation requiert une étude approfondie de chaque système. Cependant, une première distinction peut se faire entre les approches d'observations centralisées ou distribuées. Une approche centralisée définit qu'une seule entité est responsable de l'observation de l'ensemble des services réutilisés. Si cette entité tombe en panne plus aucune défaillance ne pourra être détectée. Au contraire d'une approche distribuée qui implique généralement que chaque service concret est lié à un observateur particulier. La probabilité de non détection de défaillance d'un service correspond à la probabilité que le service et son observateur tombent en panne simultanément. Cette distinction peut faire une première classification entre approches d'observations du contexte.

Nous posons $P_{ndetect}$ cette probabilité de non détection d'une défaillance. Si un composite est incapable de détecter les défaillances, la probabilité est égale à 1.

Ainsi, en appliquant la formule du risque de départ (3.1) qui combine A, B et C, on obtient :

$$Couplage = [3.5] * [3.8] * P_{ndetect} \tag{3.10}$$

$$Couplage = \{(a),(b),(c)\} * \left(\left(\prod_{k=1}^{\lambda} \prod_{i=1}^{N} \sum_{j=1}^{i} ((P_s)_{kij})^{\alpha_{kij}} \right) * C_{phys} \right) * P_{ndetect} \tag{3.11}$$

Cette formule combine les trois couplages, sémantique, syntaxique et physique. Elle évalue le couplage d'un service abstrait cible dans le contexte d'un composite particulier. Le couplage global correspond à la somme des couplages calculés pour tous les services abstraits de ce composite. Plus ce résultat est bas, plus le couplage est faible.

Notre formule du couplage se focalise principalement sur les relations entre le service composite et les services constituants, au design-time (les services abstraits) et au runtime (les services concrets). À la somme des couplages des services abstraits doit donc s'ajouter la probabilité P_{sys} d'une défaillance du côté du service composite, c'est-à-dire de l'ensemble du système qui assure les collaborations entre les services réutilisés. De fait, si c'est ce système de gestion qui tombe en panne alors plus aucune fonctionnalité métier définie par l'architecte ne pourra être assurée.

Ainsi on obtient la formule du couplage global d'un service composite cible :

$$CouplageGlobal = \Big(\sum_{k=1}^{n} Couplage_{ServiceAbstrait\_k)} \Big) + P_{sys} \tag{3.12}$$

L'évaluation de la probabilité P_{sys} dépend directement des choix d'implémentation et donc requiert une étude approfondie du système qui doit être effectuée par son développeur.

Cependant, une possibilité d'évaluation de P_{sys} serait de réutiliser notre méta-modèle de service composite et la formule du couplage globale 3.11. De fait, le méta-modèle est construit comme une abstraction des aspects importants d'une composition de services qui impactent sur le couplage. En le réutilisant comme l'expression des besoins où chaque service gestionnaire définit un service abstrait et en considérant l'architecture du système de composition à évaluer comme les services concrets sélectionnés par rapport à ces besoins, notre formule d'évaluation globale du couplage qui repose sur les services abstraits et les services concrets peut être alors appliquée.

En résumé, notre formule d'évaluation globale permet de mesurer directement le couplage d'un composite cible ce qui autorise les comparaisons entre différentes instances particulières suivant ce critère. Ce résultat est fortement dépendant du contexte courant d'un système car il se focalise sur l'étude des instances. En effet, les variables telles que le nombre de solutions alternatives sont liées à la disponibilité des services concrets à l'instant t de la mesure.

En prenant les instances de composition pour cible et en incluant des éléments du contexte, notre formule du couplage se limite uniquement à une comparaison indirecte des approches de composition de services en se concentrant sur le résultat de leur utilisation. La section suivante présente donc un cadre de comparaison directe qui s'abstrait du contexte d'un composite particulier et ainsi permet une évaluation du potentiel de réduction du couplage proposée par une approche de composition.

3.5 Comparer des approches de composition

La formule d'évaluation précédente combine les couplages sémantique, syntaxique et physique. Les valeurs retournées par ces trois couplages sont dépendantes du composite cible. Pour déterminer un cadre de comparaison indépendant de tout contexte d'un composite ou d'une application particulière, nous cherchons à exhiber les caractéristiques

des méthodes de composition dynamiques qui influencent les résultats de ces trois couplages.

3.5.1 Critères de comparaison

La spécification des critères de comparaison passe par l'étude individuelle des couplages sémantique, syntaxique et physique.

3.5.1.1 Abstraction du couplage sémantique

Le couplage sémantique (section 3.3.2) mesure les dépendances entre les services abstraits et le service composite à travers l'étude des fonctionnalités composites auxquelles ils participent. Cette mesure repose uniquement sur l'identification par l'architecte des fonctionnalités essentielles de son composite. Cependant, cette identification est subjective. Elle est strictement dépendante du domaine d'application du composite et de l'expertise de l'architecte sur son fonctionnement.

Ainsi, les caractéristiques de l'approche de composition choisie n'a pas d'influence sur le couplage sémantique.

3.5.1.2 Abstraction du couplage syntaxique

La mesure du couplage syntaxique (section 3.3.3) repose sur le nombre de solutions concrètes alternatives pour la réalisation des services abstraits. Indépendamment du contexte, le nombre de ces solutions exploitables par le service composite dépend de deux éléments : les *contraintes de sélection* et l'*algorithme de découverte de services*.

Les contraintes de sélection réduisent le champ des possibilités aux services que cherche l'architecte. L'algorithme de découverte de services récupère ces informations et modifie ce champ des possibilités en fonction de ce qu'il est capable de trouver.

Dans notre exemple, le constructeur de voitures qui cherche à identifier son futur site de production préférera une région qui peut facilement être approvisionnée par un plus grand potentiel d'industries sous-traitantes. Cet aspect représente l'algorithme de découvertes de services utilisé. De plus, si le constructeur est capable d'adapter n'importe quels types de pièces dans sa voiture (tracteur, moteur d'avion, etc.) alors un plus grand nombre d'entreprises pourront remplir ses appels d'offres. Cet aspect illustre les contraintes de sélection fournies par l'architecte.

Contraintes de sélection :

Les contraintes de sélection sont établies par les services abstraits, décrits par l'architecte qui exprime formellement ses besoins. Ces contraintes peuvent être classées en deux groupes suivant notre analyse des hétérogénéités sur les services (section 1.4.2) :

- *contraintes métiers* : elles regroupent les éléments liés au service attendu en termes fonctionnels et non fonctionnels. Elles spécifient donc les fonctionnalités recherchées par l'architecte, la qualité de service, la sécurité, la fiabilité, etc. De plus, ces contraintes expriment tous types de préférences clients telles que la localisation du fournisseur, la langue de travail, etc.

– *contraintes de réalisation* : elles représentent les implications techniques pour remplir les fonctionnalités attendues. Ces contraintes peuvent être regroupées en trois groupes : les *aspects technologiques*, les *aspects conversationnels*, et les *aspects données* (section 3.2.1).

Les contraintes métiers expriment directement l'expertise de l'architecte qui spécifie son service composite. Elles ne sont donc pas pertinentes pour une comparaison entre approches de composition car étant liées aux contextes particuliers du composite. À l'opposée, la prise en charge des contraintes de réalisation est liée aux solutions techniques employées. Ces contraintes sont donc importantes pour l'élaboration de notre cadre de comparaison.

Les trois catégories de contraintes de réalisation sont directement issues de l'étude sur la gestion des hétérogénéités présentée dans les chapitres précédents (section 1.4.2 et section 2.2.4). En effet, la gestion des hétérogénéités a pour but principal d'offrir au système la capacité d'exploiter tous types de services concrets qui répondent aux contraintes fonctionnelles et non fonctionnelles attendues par l'architecte, sans se soucier des contraintes de réalisation. L'objectif est d'offrir une gestion transparente de ces contraintes de réalisation et donc d'augmenter le potentiel de services concrets candidats.

De fait, le nombre de types d'hétérogénéités que peut gérer un système influence directement les contraintes de sélections qu'il pose. Chaque hétérogénéité gérable est une contrainte de sélection en moins sur les services concrets. Ainsi, plus ce nombre d'hétérogénéités gérées est élevé, plus le potentiel de services candidats exploitables est important. La mesure du couplage syntaxique sur un composite issu de ce système est potentiellement plus faible. Ainsi, une classification hiérarchique des approches de composition peut être faite en fonction du nombre d'hétérogénéités gérées suivant les trois groupes des contraintes de réalisation.

Algorithme de découvertes de services :

L'autre variable qui influence le potentiel de services concrets exploitables est l'algorithme de découverte utilisé. Nous classifions les algorithmes existants suivant trois caractéristiques :

– le type de correspondances supporté, 1-1 ou 1-N ;
– la flexibilité de la correspondance ; et
– la gestion des hétérogénéités suivant les aspects technologiques, conversationnels, et de données.

Le premier niveau de classification porte sur le type de correspondance : 1-1 [GNY04, VGS+05] ou 1-N [KKS07]. L'approche 1-N inclut la correspondance 1-1 et donc offre statistiquement un plus grand potentiel de résultats. Par extension, une approche capable d'effectuer des correspondances M-N aurait un potentiel encore plus important (annexe B).

Une approche 1-N identifie une composition de services concrets aux besoins exprimés par un service abstrait. Ces besoins correspondent à un ensemble de contraintes métiers et de contraintes de réalisation. À partir de là, nous classons les solutions concrètes apportées par cette approche 1-N en trois catégories :

– *cas a* : un unique service concret est identifié pour remplir les contraintes métiers

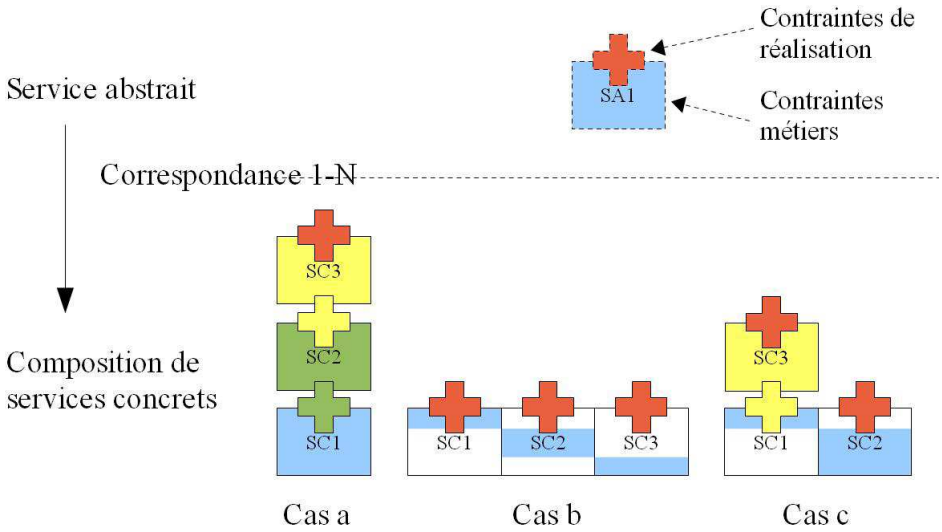

Fig. 3.8 – Natures des correspondances 1-N

du service abstrait. Cependant, ce service concret ne respecte pas les contraintes de réalisation et l'algorithme 1-N cherche à sélectionner et composer d'autres services concrets capables d'assurer ces contraintes de réalisation. Par exemple, dans la figure 3.8 cas a, deux services concrets, SC2 et SC3, sont nécessaires pour assurer les contraintes de réalisation de SC1. SC2 et SC3 agissent comme des médiateurs sur les contraintes de réalisation entre SC1 et le service abstrait recherché SA1.

- *cas b* : les contraintes métiers sont réparties sur N services concrets en collaboration qui respectent tous les contraintes de réalisation. Aucun travail de support intermédiaire n'a été effectué pour assurer ces contraintes de réalisation. Par exemple, dans la figure 3.8 cas b, les contraintes métiers sont réparties entre trois services concrets, SC1, SC2 et SC3.
- *cas c* : correspond à une combinaison des deux possibilités précédentes avec une répartition des contraintes métiers et des contraintes de réalisation. Par exemple, dans la figure 3.8 cas c, les contraintes métiers sont réparties entre deux services concrets, SC1 et SC2, et un service concret SC3 est utilisé pour assurer les contraintes de réalisation de SC1.

Un second niveau de classification porte sur la flexibilité de la correspondance, c'est-à-dire la capacité de l'algorithme de découverte de services à proposer des solutions concrètes qui ne respectent pas exactement les contraintes posées, mais qui pourraient répondre aux besoins de l'architecte [KFS06]. Nous distinguons les correspondances fortes et les correspondances faibles.

- *correspondances fortes* : représentent des correspondances exactes entre les besoins exprimés par les services abstraits et les services concrets. Par exemple, dans le cas d'une correspondance 1-1, un service abstrait définit un ensemble de contraintes qui sont toutes remplies par un unique service concret. Une correspondance forte est l'approche de base de tout algorithme de découverte.

– *correspondances faibles* : représentent la capacité à découvrir des solutions non exactes mais qui peuvent intéresser l'utilisateur. Ces solutions peuvent correspondre à des services concrets qui fournissent plus de fonctions que celles requises par les services abstraits, ou inversement, des services concrets qui ne les remplissent pas toutes [KFS06, LL08, ZMH09].

Les correspondances fortes et faibles peuvent intervenir sur des correspondances de types 1-1 ou 1-N.

Un dernier niveau de classification porte sur la capacité de l'algorithme de découvertes à gérer l'hétérogénéité des contraintes de réalisation. En effet, certains algorithmes encapsulent cette gestion des hétérogénéités dans leurs résultats de façon transparente, c'est-à-dire qu'ils sont capables de réduire la distance entre services abstraits et services concrets en s'abstrayant de certaines contraintes de réalisation.

Typiquement, ces algorithmes de découvertes sont de deux natures :

– par génération de ce que nous nommons des *médiateurs* [LDT07, ZMH09, DGD09], c'est-à-dire des entités logicielles spécifiées pour réduire la distance entre services abstraits et services concrets afin de compenser les incompatibilités de réalisation. Le résultat retourné est un service candidat qui inclut une composition du service concret et de son médiateur. L'objectif est de s'assurer que la composition de l'architecte puisse exploiter directement le service concret sans modification sur les contraintes de réalisation qu'il a exprimées via son service abstrait. Le résultat par génération de médiateur peut s'apparenter à une correspondance 1-2.
– par découvertes de services concrets dédiés à la réduction des contraintes de réalisation. Cette approche correspond au cas a de la figure 3.8 pour les correspondances 1-N.

Ainsi, les méthodes mises en place peuvent être incorporées dans la catégories des algorithmes de type 1-N. Cependant, nous avons explicitement extrait la gestion des hétérogénéités afin de mettre en évidence l'ensemble des méthodes spécifiques que l'on peut rencontrer dans les approches de découvertes de services.

En résumé, les critères de comparaison liés au couplage syntaxique qui ont été retenus sont :

– la *gestion des hétérogénéités* : les types d'hétérogénéités supportés, répartis suivant les trois catégories *technologique*, *conversationnelle* et *de données*. Dans un souci de lisibilité, ces catégories regroupent de façon transparente les aspects pris en compte soit par le moteur de composition, soit par le moteur de découverte de services ;
– le *type d'actions d'adaptations* :le type de correspondances supportées par le moteur de découverte de services répartis suivant trois catégories 1-1, 1-N et M-N.
– la *flexibilité de correspondance* : forte ou faible

3.5.1.3 Abstraction du couplage physique

Les métriques liées au couplage physique se focalisent principalement sur l'étude des communications en prenant en compte des éléments tels que les appels de méthodes, les nombres de messages échangés, la complexité de ces messages, etc.

Ces éléments sont fortement dépendants de l'application et des fonctionnalités métiers définies par l'architecte. Cependant, ils sont aussi influencés par les choix technologiques qui supporteront les communications entre les services concrets sélectionnés. Les approches synchrones ou asynchrones, les protocoles de gestion des messages et des transactions, les approches de communication par notification d'évènements sont autant de paramètres à prendre en compte pour déterminer le couplage physique et requiert une analyse poussée qui n'a pas été développée dans cette thèse car se trouvant hors de son domaine d'étude.

Cependant, nous pouvons faire une distinction à un plus haut niveau directement liée aux principes SOSE sur le *type de schéma de collaboration* entre chorégraphie et orchestration.

Un schéma de collaboration exprime les différents flots de travail et de données pour assurer la coordination entre les services concrets utilisés. La chorégraphie repose sur des communications horizontales, directement entre les services concrets. Au contraire, l'orchestration impose des communications uniquement verticales entre les services concrets et le service composite qui centralise les échanges. Une approche orientée uniquement vers la réduction du couplage cherche à minimiser les communications transversales entre les services concrets d'un même niveau de description. L'approche par orchestration offre naturellement une meilleure indépendance entre les services réutilisés. Ainsi, l'ensemble des aspects liés au couplage physique est centralisé uniquement entre le service composite et ses services concrets. À partir de là, pour réduire encore le couplage il faut agir sur cette relation entre le composite et ses constituants.

3.5.1.4 Abstraction de la formule d'évaluation globale du couplage

La formule d'évaluation globale du couplage fait intervenir trois opérandes : criticité, probabilité d'occurrence de défaillances, et probabilité de non détection de ces défaillances. Les informations des couplages sémantique, syntaxique et physique sont réparties à l'intérieur de ces opérandes et nous permettent de définir les deux formules résultats : 3.11 pour l'évaluation d'un service abstrait, et 3.12 pour l'évaluation du service composite dans son ensemble.

La formule 3.11 fait intervenir la probabilité de non détection d'une défaillance. Cette probabilité ne peut être déterminée qu'à travers une étude poussée du *système d'observation du contexte* mis en place dans le service composite cible (dans notre méta-modèle de service composite, ce système est représenté par les différents moniteurs). Cette étude doit être personnalisée à chaque système et requiert l'expertise de son concepteur. Elle se trouve donc hors du cadre de notre thèse. Cependant, comme présenté dans la section 3.4.3, nous pouvons faire une première distinction haut niveau entre les approches d'observation du contexte *centralisées* ou *distribuées*.

De son côté, la formule 3.12 fait intervenir la probabilité de défaillance du *système d'exécution de la composition de services* (représenté dans notre méta-modèle de service comme le GSC et les différents services gestionnaires). De la même façon que pour la probabilité de non détection d'une défaillance, elle nécessite une étude approfondie et personnalisée du système considéré. Cependant, comme expliqué dans la section 3.4.3,

le méta-modèle de service composite et la formule 3.11 d'évaluation globale du couplage peuvent être réutilisés pour évaluer le couplage d'un système de composition de services.

3.5.2 Cadre de comparaison

Le tableau 3.2 résume nos critères de comparaison extraits des différents couplages et de la formule globale. Puis, il compare des approches existantes de compositions de services (nous reprenons certains travaux présentés dans le chapitre 1). Ainsi, nous mettons en évidence l'absence d'un modèle unique qui combine les différents aspects importants dans la réduction du couplage et démontre l'intérêt de notre proposition d'un méta-modèle de service composite (chapitre 2). De par son principe de réification des caractéristiques des approches existantes, il correspond à la somme de leurs avantages sur le couplage faible. Ce constat est évidemment à reconsidérer du fait qu'il est à l'heure actuelle une proposition de modélisation alors que les travaux dont il cherche à coordonner les propriétés sont en général associés à des outils ou des prototypes implémentés.

Tab. 3.2 – Cadre de comparaison d'approches de composition de services

	Abstraction du couplage syntaxique					**Abstraction du couplage physique**	**Abstraction de la formule globale**	
	Gestion des hétérogénéités			**Type actions adaptations**	**Flexibilité de la correspondance**	**Type de schéma de collaboration**	**Approche de gestion de composition**	**Approche d'observation du contexte**
	technologiques	conversationnels	de données					
[OAS09]	X					orchestration chorégraphie	NR : non renseigné	
[CCG+09]				1-1	Faible	orchestration	NR	centralisé
[CNP09]		X	X	1-1	Forte	orchestration	NR	centralisée
[BGPT09]				1-1	Forte	orchestration	NR	distribué
[HSD10]				1-N	Forte	orchestration	NR	centralisée
[HkO10a]	X	X	X	1-N	Forte/Faible	orchestration	NR	distribué

Comme nous l'avons précisé précédemment, la colonne *Approche de gestion de composition* fait intervenir la probabilité de défaillance du système d'exécution de la composition de services et nécessite donc une étude approfondie et personnalisée du système considéré qui n'est disponible pour aucune des approches.

Cependant, en considérant un système de gestion de composition cible comme une réalisation concrète des services abstraits (les différents services gestionnaires) exprimés par notre méta-modèle de service composite, nous pouvons réutiliser notre formule d'évaluation globale et ainsi mesurer le couplage interne à ce système. Cette approche met en évidence une lacune présente, à notre connaissance, sur l'ensemble des propositions existantes de composition dynamique. En effet, aucune d'entre elles n'est capable de modifier son système de gestion de composition en cas de défaillance. Le méta-modèle de service composite qui exprime les services abstraits ne possède pour chacun d'eux qu'une seule solution concrète de réalisation, celle proposée par le système cible. Ce dernier ne possède pas de mécanisme d'adaptation sur ces propres éléments de gestion. Par exemple, si le moteur d'exécution de la collaboration tombe en panne, le système est incapable de le remplacer. Le couplage syntaxique qui cible les dépendances entre services abstraits (les gestionnaires) et services concrets est à son maximum, c'est-à-dire fort pour l'ensemble des services abstraits.

Ce même constat peut être fait sur notre méta-modèle de service composite où le processus d'auto-adaptation (section 2.3.3) ne cible que les services métiers et non les services gestionnaires. Cependant, nous avons considéré dès le départ les différents éléments architecturaux du service composite comme des services à part entière qui peuvent donc être découverts. Le GSC[1] organise ses services gestionnaires par orchestration et ces derniers peuvent donc être facilement remplacés. Dans cette approche, le noyau dur d'un service composite serait uniquement sa gestion de l'auto-adaptation (le GSC) et les différents types de schéma de collaboration qui expriment ses fonctionnalités métiers offertes.

3.6 Conclusion

Dans ce chapitre, nous avons présenté notre vision du couplage faible dans une composition de services. L'objectif recherché est de fournir les moyens de mesurer précisément les dépendances entre services en collaboration dans une même composition. En apportant une évaluation numérique objective, nous voulons faciliter les comparaisons entre compositions de services afin de déterminer leurs bénéfices respectifs sur le couplage. De plus, cette étude apporte le socle théorique nécessaire à la définition d'un cadre de comparaison haut-niveau qui, cette fois, cible directement les approches de compositions de services. En effet, ce cadre de comparaison complète le travail précédent d'évaluation des compositions de services particulières par une comparaison entre les méthodes qui servent à produire ces compositions.

[1]Gestionnaire de service composite (section 2.2.2)

Pour remplir ces objectifs, nous proposons dans un premier temps une nouvelle définition théorique du couplage faible qui repose sur une décomposition en trois couplages distincts :

- le *couplage sémantique* : se focalise sur les dépendances au niveau des services abstraits. Il exprime l'expertise de l'architecte sur le domaine d'application de son service composite ;
- le *couplage syntaxique* : se focalise sur les dépendances entre les services abstraits et les services concrets. Il mesure les possibilités de solutions alternatives à la réalisation des besoins du composite ;
- le *couplage physique* : se focalise sur les dépendances au niveau des services concrets qui ont été sélectionnés et utilisés pour réaliser le service composite. Il repose directement sur les travaux existants qui manipulent les variables de communication.

Chacun de ces couplages est associé à une méthode de mesure. Ces mesures sont ensuite combinées dans une formule globale d'évaluation qui calcule le couplage d'une composition de services particulière. Enfin, une dernière partie aborde la construction d'un premier cadre de comparaison sur le critère de couplage entre approches de composition qui est dérivé de notre formule.

Nos travaux présentés dans ce chapitre et dans le chapitre précédent, autour des concepts de service composite et de couplage faible, sont partis d'une volonté de clarification de leur définition. En effet, l'étude bibliographique nous a permis de mettre en évidence le manque de formalisme dans leur spécification. Nous avons donc voulu combler ce manque et apporter une meilleur compréhension des notions qui y sont associées.

Au cours de notre étude, nous nous sommes rapidement rendus compte de la forte similarité entre le paradigme service et le paradigme composant (CBSE[2] [HC01, Szy02]). De fait, CBSE et SOSE reposent sur le même principe de construction de nouveaux systèmes à partir d'entités logicielles existantes, appellées composants ou services. Ils font donc face aux mêmes challenges : d'identifications de ces entités, de leurs compositions correctes pour définir les applications, d'adaptations dynamiques de ces compositions pour améliorer leur flexibilité aux changements de contextes et de besoins, et ainsi de suite. Nous étendons donc notre approche de clarification et d'explicitation pour identifier de façon plus précise les frontières entre un composant et un service. Le chapitre suivant fait un retour d'analyse sur l'état de l'art pour offrir une meilleure compréhension des similarités et des particularités des paradigmes CBSE et SOSE. Ce travail correspond au troisième et dernier axe de notre contribution.

[2]Component-Based Software Engineering

Chapitre 4

CBSE vs SOSE : vers un cadre conceptuel de comparaison

4.1 Introduction

L'ingénierie logicielle à base de composants (Component-based software engineering - CBSE [HC01, Szy02]) et l'ingénierie logicielle orientée services (Service-oriented software engineering - SOSE [SD05, The08]) sont deux domaines établis du génie logiciel.

Le CBSE s'inspire des autres disciplines de l'ingénierie telles que l'ingénierie mécanique ou électrique. Il repose sur l'idée de "*construire des systèmes à partir de composants*". Ces dernières années ont été témoins de l'important succès des approches basées composants dans le développement d'applications variées telles que les systèmes distribués ou basés web, les applications graphiques ou le domaine des systèmes embarqués. Le CBSE est motivé par la réduction du temps de mise sur le marché de nouvelles applications en facilitant de façon significative la réutilisabilité et la gestion de sa complexité.

Le SOSE se base sur le modèle de mondialisation des entreprises modernes et la notion associée de service. Le principe de la mondialisation est de sous-traiter les fonctions à faible valeur ajoutée ou échappant au domaine de compétence de l'entreprise afin de se concentrer sur les aspects innovants du système. Ce sont ces aspects innovants qui font la particularité de l'entreprise et sa valeur ajoutée par rapport aux autres. La sous-traitance représente les services réutilisés. L'objectif est de chercher à réduire les coûts de l'exploitation de ces services en jouant sur la concurrence. L'origine de l'orienté

service vient des demandes grandissantes pour des applications devant supporter des environnements hautement volatiles et hétérogènes. Le SOSE est motivé par la réduction du temps de production en réutilisant des fonctionnalités existantes fournies par des tiers.

Ainsi CBSE et SOSE ont la réutilisation d'entités logicielles existantes, composants ou services, comme socle commun. Tous deux reposent sur le concept d'architecture logicielle [MT00] dans lequel un système est vu comme une structure claire d'entités et de relations entre ces entités. Le SOSE est un paradigme plus récent dont l'approche est influencée par le CBSE, d'une façon similaire à l'incidence que l'objet a eu sur le composant. De plus, ces dernières années ont vu CBSE et SOSE coexister tout en ayant un développement parallèle. Cela a entraîné l'apparition de concepts similaires ou spécialisés qui sont souvent mis en juxtaposition avec des confusions de vocabulaire. Ces confusions sont accentuées par l'existence d'approches hybrides qui empruntent des éléments aux deux paradigmes. Enfin, les applications modernes qui combinent composants et services accentuent cet entrelacement ambiant et la compréhension globale devient plus difficile.

Notre objectif est de clarifier les frontières entre CBSE et SOSE. Dans ce chapitre, nous proposons un premier cadre conceptuel de comparaison basé sur une approche descendante "*top-down*". Le principe d'une approche top-down est de se concentrer sur la différenciation des aspects conceptuels directement liés aux paradigmes, par opposition à une approche "*bottom-up*" qui compare les différentes technologies d'implémentation de ces paradigmes. Notre cadre conceptuel de comparaison repose sur un triptyque produit, processus et qualité qui organise les caractéristiques respectives de CBSE et SOSE. Le but est d'offrir aux architectes une meilleure compréhension des implications et des conséquences du choix de l'un ou l'autre de ces paradigmes. De plus, ce cadre de comparaison prend en compte l'orienté objet (OO [Oa99]) afin d'apporter une vision globale de l'évolution des préoccupations du génie logiciel entre objet, composant et service. Nous cherchons à complèter le travail effectué autour de la différenciation entre objet et composant [Szy02, Oa05] en abordant celle entre composant et service.

Ce chapitre s'organise de la façon suivante. La section 4.2 discute les origines des confusions entre les approches composant et service. Elle pose les limites des tentatives existantes d'explicitation de ces confusions et dresse les objectifs que nous nous sommes fixés. Nous présentons dans la section 4.3 les différences conceptuelles qui sont à l'origine de la définition des paradigmes composants et service. Dans la section 4.4, nous nous focalisons sur l'aspect quantitatif de notre cadre de comparaison. Cet aspect classifie et compare les éléments relatifs aux notions de produit et de processus. La section 4.5 aborde l'aspect qualitatif du cadre de comparaison et mesure les bénéfices respectifs des différents paradigmes. Après avoir défini les éléments du cadre conceptuel et proposé notre comparaison entre OO, CBSE et SOSE, nous le réutilisons dans la section 4.6 pour déterminer les aspects du SOSE qui sont améliorés par nos travaux sur le méta-modèle de service composite et sur le couplage faible. Enfin la section 4.7 revient sur les contributions de ce travail et conclut ce chapitre.

4.2 Analyse de l'existant

4.2.1 Origines des confusions

Les origines des confusions entre CBSE et SOSE sont nombreuses et représentent autant de justifications à ce travail. Nous avons choisi de regrouper ses confusions suivant deux points de vue :

- les *confusions conceptuelles* : regroupent les éléments de confusion issus des racines conceptuelles communes entre les composants et les services ; et
- les *confusions techniques* : mettent l'accent sur les confusions de vocabulaire entre l'orienté composant et l'orienté service. Ces confusions sont renforcées par des approches, conceptuelles ou technologiques, qui empruntent des notions et des mécanismes issus des deux paradigmes ;

4.2.1.1 Confusions conceptuelles

Comme présenté dans le chapitre 1, le paradigme service est relativement récent. Chronologiquement, il s'est développé après le paradigme composant. De ce fait, la communauté service s'est naturellement inspirée de la recherche autour des composants, de la même façon que la communauté composant s'est construite sur les années d'expertise issues des objets. Un certain nombre de concepts s'est donc transmis d'un paradigme à l'autre. L'identification de ces héritages est un point crucial pour mieux appréhender les confusions entre CBSE et SOSE, et ainsi améliorer leur compréhension respective.

Composant et service reposent sur un principe de réutilisation qui est aussi à l'origine du développement du paradigme objet. En effet, l'idée directrice de l'orienté objet est de limiter les répétitions de codes en proposant des concepts et des mécanismes qui peuvent supporter la sauvegarde de l'expérience acquise et garantir par la suite son exploitation. Cette réutilisation pose la base du développement *for reuse* et *by reuse* [Oa99, Oa05].

- **for reuse** : processus de développement d'entité logicielle dont le but est d'être réutilisée. Il implique la définition d'éléments spécifiques pour faciliter cette réutilisation.
- **by reuse** : processus de développement d'entité logicielle à partir d'entités existantes. Il implique la formalisation de méthodes particulières d'exploitations de ces entités existantes tout en garantissant la sûreté de leurs réutilisations.

En orienté objet, les développements for reuse et by reuse sont supportés par la *classe*. La *classe* permet d'encapsuler données et fonctionnalités liées à l'implémentation de l'expérience que veut transmettre le développeur. Elle assure la conservation et la réutilisation à l'identique de cette expérience. De plus, la classe supporte le processus très puissant d'*héritage* qui permet de créer des nouvelles classes à partir de celle héritée. Ces nouvelles classes font évoluer les éléments encapsulés de façon à spécialiser leur comportement en fonction des besoins.

CBSE et SOSE travaillent dans le prolongement de cette pensée autour de la réutilisation et des notions associées de for reuse et by reuse. Ils cherchent à développer et à améliorer la couche d'encapsulation afin de faciliter l'exploitation des ressources

sous-jacentes et supporter les nouvelles exigences liées à l'avènement d'Internet et des applications distribuées. L'objectif est de simplifier leur utilisation par des tiers en limitant les efforts d'apprentissage et d'adaptation, tout en augmentant la granularité de ces ressources. De fait, CBSE et SOSE renforcent les principes for reuse et by reuse en apportant une nouvelle formalisation des contraintes de communications entre entités logicielles réutilisées. Cette formalisation améliore les traitements automatiques (assemblage, vérification de la correction, etc.) pour des entités à plus forte granularité que les objets. Une plus forte granularité correspond à des ressources encapsulées plus importantes et nécessitant des interactions plus complexes pour les exploiter.

L'avènement des composants, puis des services à sa suite, a coïncidé avec un glissement du focus de la communauté du génie logiciel où la problématique première des objets, qu'est la *réutilisabilité*, laisse sa place à la *composabilité*. La composabilité correspond à la propriété d'une entité logicielle à être composée et à la facilité de cette composition. Composant et service standardisent les descriptions de l'interface d'accès aux ressources afin d'appporter cette composabilité et de faciliter la définition des collaborations entre ces ressources. Tout système basé composants ou services se voit décrit comme un ensemble d'entités logicielles clairement définies dans les fonctionnalités qu'elles offrent, et de relations de communication explicitées entre ces entités de manière à diriger leurs collaborations. Ainsi, CBSE et SOSE partagent, en plus de leur racine objet, cette structuration des systèmes plus connue sous le terme d'Architecture logicielle [MT00].

4.2.1.2 Confusions techniques

Les confusions techniques sont directement issues de l'évolution chronologique, des objets aux composants, puis aux services, où chaque communauté s'est inspirée de l'expérience des approches plus anciennes pour construire et développer sa propre interprétation du développement logiciel.

Ce fait est particulièrement marqué entre composant et service de par leur rapprochement temporelle. En effet, les recherches dans le CBSE et le SOSE cohabitent depuis une dizaine d'années maintenant dans le génie logiciel. Ils ont suivi et suivent encore un développement parallèle en fonction de leurs cibles d'applications différentes. Ce développement se mêle aux influences mutuelles du fait de la proximité des problématiques, en particulier sur la composition. Il en découle des confusions de vocabulaires où se confondent des concepts communs sous des terminologies différentes, et inversement des terminologies similaires pour des concepts différents [BL07]. L'exemple le plus significatif est la notion de service qui est probablement l'un des mots les plus galvaudés de ces dernières années qui peut se référer à tour de rôle aux interfaces fournies et requises des composants, à des activités d'un processus métier, aux fonctionnalités d'un système, au système en lui-même, à des ressources physiques, etc. Ainsi, sa maîtrise nécessite de jongler entre niveaux de granularité, perspectives haut niveau et bas niveau, paradigmes de développement et ainsi de suite.

Les communautés service et composant sont très proches et coïncident par bon nombres de leurs membres qui sont à cheval sur les deux paradigmes. L'illustration parfaite

est le développement d'approches que l'on peut qualifier d'hybrides car empruntant et combinant des éléments conceptuels ou techniques issus du CBSE et du SOSE.

SCA [OAS09] (Service Component Architecture) est, de notre point de vue, un exemple de ces approches hybrides. SCA combine architecture à composants pour la conception de nouveaux systèmes et des capacités de gestion des hétérogénéités et de la dynamicité nécessaire à l'orienté service. De fait, sa brique architecturale de base, le composant SCA, tente d'atteindre le même degré d'encapsulation qu'un service dans sa capacité à gérer des ressources hétérogènes. L'outil actuel [Ecl09] permet de faire communiquer, simplement et de façon transparente aux développeurs, des composants SCA implémentés dans un certain nombre de technologies différentes telles que Java, EJB, services web [ACKM03], etc. Évidement, le nombre des technologies supportées est limité mais il illustre cette approche typiquement service de collaboration transparente entre ressources hétérogènes. Enfin, les versions en cours des outils SCA apportent une dynamicité accrue dans l'établissement des connexions entre composants SCA, et entre les composants SCA et leur implémentation. Cette dynamicité s'inscrit à son tour dans une approche SOSE.

Ainsi, SCA illustre toute la complexité des entrelacements entre les composants et services où il est présenté comme une solution simple à l'implémentation de systèmes oriéntés services et repose sur une architecture à base de composants où l'interface fournie est elle-même qualifiée de service.

FROGi [DCD06] est un autre modèle hybride. Il est une extension du modèle de composant Fractal qui supporte des caractéristiques additionnelles. FROGi est implémenté sur la plate-forme service OSGi [OSG11] en combinant Julia, l'implémentation de référence de Fractal basée Java. Il supporte une caractéristique primordiale de l'approche service qu'est la découverte, la sélection et la composition de services au runtime. Ces services sont à nouveau identifiés comme des interfaces fournies des composants Fractal.

De plus, l'ensemble des confusions techniques présentées sont renforcées par une réalité simple où les approches d'implémentation classiques voient les services pouvant être développés par des composants, de la même façon que les composants sont supportés par des objets.

4.2.2 Limitations des comparaisons existantes

CBSE et SOSE sont deux domaines majeurs du génie logiciel. Il existe donc un grand nombre de travaux qui se focalisent sur leurs spécifications et un nombre encore plus important de propositions conceptuelles ou technologiques qui cherchent à les réaliser. La multitude de ces travaux a naturellement induit la nécessité d'études comparatives afin d'aider la communauté à cartographier la recherche actuelle et à en identifier les limites et les possibilités d'améliorations.

4.2.2.1 Comparaisons intra-paradigmes

Différents travaux de comparaison ont été proposés dans chacun des paradigmes. Nous pouvons citer [CCSV07] pour le CBSE et [BKM07] pour le SOSE comme les plus représentatifs et les plus complets. Cependant, un grand nombre de travaux de recherche existe sur la comparaison directe entre différentes technologies telles que [MG09] pour la différence entre les implémentations services basées SOAP [SR09] ou REST [Fie00], [TWKI08] sur les méthodes de composition de services web, etc.

De par la restriction de leur analyse à un paradigme particulier, leurs auteurs ont naturellement opté pour des approches bottom-up. En effet, dans une approche bottom-up, les travaux se focalisent d'abord sur les technologies en comparant les mécanismes et les caractéristiques proposés par les différentes réalisations. Puis, ils font l'analyse des conséquences sur les qualités, d'une part de la technique en elle-même et puis d'autre part, des résultats de son application. Cette analyse qualitative permet aux utilisateurs d'identifier de quelle manière et en quelle proportion les technologies comparées s'inscrivent dans le paradigme de développement étudié. Bien qu'extrêmement utiles pour la compréhension d'un paradigme particulier, ces approches ne permettent pas d'effectuer des comparaisons conceptuelles directes entre CBSE et SOSE car elles définissent un ensemble de cadres spécifiquement adaptés à leur paradigme respectif. L'emploi des mêmes critères d'un cadre orienté pour la comparaison de modèles de composants afin d'évaluer une technologie service biaiserait la lecture finale en ne prenant pas en compte certains aspects qui sont spécifiques à cette dernière.

4.2.2.2 Comparaisons inter-paradigmes

La plupart des articles de recherche qui abordent conjointement les domaines des composants et des services suivent cette même approche bottom-up [CCSV07, BKM07]. Ils peuvent être regroupés suivant deux préoccupations :

- combiner certaines technologies CBSE et SOSE afin de réaliser des applications particulières ;
- comparer deux à deux certaines technologies composant et service.

Combinaisons de technologies CBSE et SOSE :

Lors de leur recherche, les travaux de combinaisons de technologies mettent en évidence certains points particuliers de dissension entre les composants et les services qu'ils ont surmontés via leur proposition. Bien qu'utiles lors de l'élaboration de notre propre travail de comparaison, leurs études des différences restent superficielles et spécifiques, et de fait elles n'abordent pas les aspects plus haut niveau sur les origines conceptuelles et les implications qualitatives qui en découlent.

Par exemple, dans [MHB09], les auteurs soulignent une différence de processus de composition lors des combinaisons de SOFA 2 [BHP07], plate-forme composant, et OSGi Service Platform, plate-forme service. En CBSE, les composants sont assemblés via des connecteurs ou du code glue, et les connexions entre ces entités sont en général statiques et explicitement décrites. En SOSE, ces connexions sont faites au runtime, lorsque les

différents fournisseurs de services sont découverts et leurs services liés. Les auteurs cherchent à composer composants et services ensemble malgré cette différence d'approche. Ils développent un mécanisme de proxy particulier entre composants SOFA et services OSGi chargé de supporter la dynamicité des connexions.

Ce mécanisme de proxy existe aussi sous la plate-forme Spring framework [Spr11] qui manipule des POJOs comme des composants et assure leur composition avec des services.

Dans une thèse [Sto05] soutenue à l'université de Delft, son auteur présente une méthode d'ingénierie logicielle à base de composants qui se combine avec les services web. Il aborde de nombreux aspects du développement CBSE à travers une grande partie du cycle de vie, des méthodes de design des composants et de l'architecture, jusqu'à l'implémentation et l'exécution de l'application. Cependant, son approche reste cantonnée à une perspective composant et à l'intégration de services web. L'auteur ne fait qu'effleurer la comparaison CBSE vs SOSE en soulignant l'importance de la publication et la découverte dynamique de services.

Nous pouvons à nouveau citer les approches hybrides FROGi [DCD06] et SCA [OAS09] qui combinent, Fractal et OSGi via l'emploi de Julia pour le premier, et technologies objet, EJB et services web pour le second.

Comparaison de technologies CBSE et SOSE :

Un ensemble d'autres études existent et comparent des technologies deux à deux issues de différents paradigmes de développement. Cependant, ces travaux restent extrêmement bas niveau et n'offrent pas le recul nécessaire à une compréhension globale des différences entre paradigmes. De plus, ils sont généralement limités à des préoccupations qualitatives ciblées telles que la performance ou la sécurité.

Par exemple, [VPT06] compare les technologies CORBA [OMG08], Enterprise JavaBeans et services web pour le développement d'applications distribuées. La méthode de comparaison est de type bottom-up où elle se focalise sur différents facteurs bas niveau, tels que la représentation des données, les protocoles de transferts, les mécanismes de découvertes ou encore la gestion de la sécurité, avant d'effectuer des mesures de performances. De la même façon, [KH06] compare les performances entre DCOM [Mic11], CORBA et les services web.

La diversité de ces travaux exprime toute la complexité de la maîtrise conjointe du CBSE et du SOSE et renforce notre objectif de départ d'explicitation de leurs différences.

Malgré leur proximité conceptuelle et les multiples confusions qui en découlent, peu de travaux se sont concentrés exclusivement sur la comparaison et l'établissement des différences entre CBSE et SOSE à un niveau conceptuel. À notre connaissance, un seul article s'intéresse exclusivement à ce problème. Dans cet article [BL07], les auteurs apportent un début de réponse à la problématique des confusions techniques en clarifiant certains éléments de vocabulaires associés à des propriétés particulières. Ils dressent une liste des concepts et des principes qu'ils définissent comme clés et qui caractérisent le CBSE et SOSE. Puis, ils effectuent une comparaison point à point entre ces éléments clés. Cette liste se divise en six catégories : *concepts*, *principes*, *processus de développement*,

technologie, *qualité* et *composition*.

Ces six catégories sont composées d'un certain nombre de sous-catégories et chacune d'elles est associée à sa spécification en composant, puis en service. Ce travail pose une base solide de définition des caractéristiques liées au CBSE et SOSE et ainsi clarifie une partie des confusions de vocabulaires. Cependant, il reste limité sous plusieurs aspects. La première limitation vient de la nature du cadre de comparaison en lui-même qui pose une liste de critères sans exprimer la façon d'exploiter ces critères et l'objectif de leur choix. La seconde limitation vient de la manière d'utiliser ces critères à travers lesquels les auteurs statuent uniquement des solutions portées par les deux paradigmes. Ces derniers ne font que lister leurs concepts à travers les différentes catégories sans effectuer de comparaisons entre eux. De fait, ce travail ne représente qu'une liste de définitions juxtaposés sans aucun jugement de valeur et analyse des conséquences nécessaires à la comparaison. Les implications au niveau des qualités des processus du développement et des systèmes produits en général sont ignorées.

D'autres recherches telles que [Att09, NGM+08] mettent en parallèle composant et service, mais ces dernières n'abordent cette problématique que via un point de vue très spécifique et les résultats qu'ils présentent ne constituent pas le sujet principal de leur publication. Leurs conclusions restent donc partielles et ne permettent pas de statuer clairement des différences entre composant et service.

4.2.3 Objectifs de ce cadre conceptuel de comparaison

L'objectif de notre cadre conceptuel de comparaison est de combler le manque autour de l'identification claire des différences entre CBSE et SOSE. Le but est d'offrir une meilleure compréhension aux utilisateurs par une synthèse comparative des deux paradigmes afin de les aider à statuer dans l'emploi de l'un ou l'autre de ces paradigmes. Cette aspiration passe par une maîtrise de leurs concepts respectifs, dans leur définition, puis dans l'analyse des conséquences sur la qualité.

Ce travail de comparaison entre composant et service suit le même effort et a la même vocation que ceux effectués autour de la comparaison entre objets et composants [Szy02, Oa05]. L'objectif commun est la synthèse et la compréhension des différences dans un unique cadre appréhendable d'un seul tenant.

Ainsi, l'approche que nous développons suit un schéma top-down qui, par opposition aux travaux bottom-up précédents, se focalise dans un premier temps aux niveaux conceptuels, directement sur les paradigmes, avant de chercher à y dériver les implications qualitatives. Ce focus à haut niveau permet de définir un cadre global capable de manipuler composant ou service. Dans cette définition du cadre de comparaison, nous cherchons à la fois : la *généralité*, la *minimalité* et la *complétude*.

- la *généralité* : dans l'identification des catégories et sous-catégories du cadre de comparaison qui ne doivent pas être dépendantes d'un paradigme particulier mais au contraire offrir un point de vue extérieur sur lequel on puisse projeter les éléments du CBSE ou du SOSE. Cette généralité permet de ne pas favoriser l'un ou l'autre des paradigmes, et aussi d'assurer la réutilisabilité du cadre qui pourra être employé

à la comparaison des différents autres paradigmes de développement. De fait, la comparaison effective CBSE vs SOSE présentée dans les sections suivantes inclut l'orienté objet (OO) afin d'apporter une vision plus globale de l'évolution des préoccupations du génie logiciel à travers ces trois paradigmes ;
- la *minimalité* : dans les catégories choisies et dans les éléments classifiés qui doivent extraire l'essence des paradigmes uniquement nécessaires à l'identification de leurs différences ; et
- la *complétude* : dans l'identification des différences qui doit permettre d'appréhender entièrement les conséquences sur la qualité du choix de l'un ou l'autre des paradigmes. Cette complétude du cadre passe par la possibilité laissée à ses utilisateurs de personnaliser l'analyse qualitative.

4.3 Différences conceptuelles

CBSE et SOSE ont une approche très similaire qui se base sur la construction de systèmes à partir d'entités logicielles existantes, composants ou services. Ils ont un objectif commun de maximisation de la réutilisabilité qui est directement issu de l'objet. CBSE et SOSE partagent le même processus de développement global qui se compose des activités d'identification des entités logicielles (composant ou service) qui répondent aux besoins, puis de combinaison de ces entités pour réaliser l'application finale. Ils reposent sur les mêmes notions de composition, pour la construction de nouvelles entités à partir de l'existant, et de composite, pour garantir une approche homogène où toute entité peut être vue comme un composant ou un service. Cette approche permet la manipulation de structures de description claires en terme de composition de composites. Ainsi, CBSE et SOSE facilitent le développement incrémental et l'exploitation de connaissances. Cependant, bien que CBSE et SOSE aient le même objectif global, les concepts derrière les notions de composants et de services sont différents.

Ainsi, nous confrontons composant et service suivant trois aspects :
- Différence d'usage et de responsabilité propriétaire
- Différence de rapport au couplage
- Différence de granularité

4.3.1 Différence d'usage et de responsabilité propriétaire

Un composant est dit *off-the-shelf* (*sur étagère*) [NGM+08,CCL06,HC01] par adoption d'une *pièce* technologique, le composant, qui est disponible pour les développeurs. Ces derniers récupèrent le bloc logiciel composant et se chargent de l'incorporer suivant leur besoin.

Un service est centré sur l'utilisation d'une *fonction* fournie par un tiers [SD05,OAS08, The08]. Un consommateur de services exploite uniquement le résultat issu de l'invocation du service cible.

Ces deux visions semblent proches au premier abord cependant elles ont un impact significatif sur la répartition des responsabilités entre fournisseur et consommateur. Pour illustrer cette distinction nous prenons un exemple de l'industrie du jeux vidéo sur PC. Cette industrie repose principalement sur deux modèles de distribution de contenus :

- modèle classique : achat d'un jeu en magasin spécialisé ou téléchargement sur internet ;
- modèle "Cloud gaming" : achat d'un abonnement pour jouer aux jeux disponibles directement sur une plate-forme internet.

Le modèle classique illustre l'approche composant. Le modèle dit de "Cloud gaming" illustre l'approche service.

4.3.1.1 Responsabilité sur un composant

Le premier modèle classique correspond à un joueur qui achète une copie de son jeu. Cette copie est récupérée soit sur un support physique, généralement un DVD, soit de façon dématérialisée via les plate-formes de téléchargements telles que STEAM[1]. Le joueur est ensuite responsable de l'installation du jeu sur sa propre machine, c'est-à-dire de son déploiement. C'est uniquement après cette installation qu'il peut lancer l'application et commencer à jouer.

Ce modèle de distribution correspond à une approche composant. Typiquement, le jeu (le composant) vient avec un manuel d'instructions (la documentation) qui définit un certain nombre de contraintes côté client. Ces contraintes sont de deux natures :

- les *contraintes de déploiement* : le fournisseur d'un jeu vidéo PC définit la configuration système minimale requise en terme de puissance de calcul (processeur, carte graphique, mémoire vive, etc.), de capacité de stockage (disque dur), de ressources sonores, etc. Le système du client doit respecter ces exigences pour être capable d'installer et d'exécuter le jeu. Le processus d'installation pose en lui-même des contraintes que ce soit sur la localisation précise sur le disque dur ou encore les besoins de connexions à internet, de clé d'authentification, etc.
  En CBSE, ces contraintes d'installations sont typiquement définies par le modèle de composant [CCSV07] choisi. Chaque modèle est associé à un environnement système particulier pour pouvoir être utilisé. De plus, ce modèle pose les règles de déploiement associées à ses composants.
- les *contraintes d'utilisation* : chaque jeu pose une liste de commandes particulières qui détermine la façon d'interagir avec lui et les successions d'actions qui sont nécessaires à la progression à travers les niveaux (gameplay, leveldesign, etc.). Ces différents éléments posent les règles à respecter si l'utilisateur veut profiter pleinement de l'expérience proposée.
  En CBSE, ces contraintes d'utilisation sont typiquement définies par l'interface contractuelle du composant. Le respect de cette interface est primordial pour garantir l'exploitation correcte des ressources suivant les possibilités préalablement déterminées par le fournisseur du composant.

[1] http ://store.steampowered.com/

4.3.1.2 Responsabilité sur un service

Le second modèle de distribution, appelé Cloud gaming, illustre la notion de service. Dans ce modèle, le joueur paie le droit de jouer à un jeu qui est exécuté sur une plate-forme à distance sous la responsabilité du fournisseur. Il a uniquement besoin de l'interface et de la connexion adéquates pour accéder à cette plate-forme. De fait, le joueur n'est plus responsable du déploiement du jeu sur sa propre machine. Les seules informations qui lui sont nécessaires sont donc comment accéder à cette plate-forme et comment jouer au jeu.

Ainsi, il n'existe plus de contraintes de déploiement liées à l'installation du jeu mais uniquement les contraintes d'utilisation. Cette absence de déploiement a plusieurs avantages. D'une part, elle simplifie l'exploitation des ressources en supprimant les efforts qui accompagnent la compréhension des phases d'installation. D'autre part, elle garantit une utilisation de ces ressources qui se veut normalement optimale. En effet, l'application est exécutée directement sur l'environnement du fournisseur. Ce dernier a donc un contrôle total de l'exécution. Ainsi, il est plus à même de garantir la qualité promise à ses clients.

Dans notre exemple, la qualité d'un jeu vidéo (fluidité, graphisme, etc.) varie en fonction du système sur lequel il est installé. En étant exécuté sur une plate-forme à distance, ce jeu a une qualité identique pour chacun des joueurs connectés. De plus, les utilisateurs qui, à l'origine, ne possédaient pas les configurations systèmes requises pourront profiter de ce service. Ainsi, les contraintes sur le client se réduisent uniquement à sa capacité de communication.

Enfin, un autre avantage significatif de ce modèle de relations service entre clients et fournisseurs est la transparence des évolutions du service tant que ces dernières ne modifient pas les contraintes d'utilisation de départ (interface de connexion, protocoles, etc.). De fait, les nouvelles versions sont directement accessibles sans nécessiter d'adaptation du côté client. Au contraire, dans une approche composant, si le client veut profiter de ces évolutions il doit les récupérer et les déployer de lui-même. Des problèmes liés à ce déploiement peuvent apparaître dans le cas où le système du client ne supporte plus le composant mis à jour.

Le Cloud gaming illustre cet avantage où différentes versions d'un même jeu peuvent se succéder de son façon transparente aux utilisateurs. La distribution classique, quant à elle, impose aux joueurs la récupération d'un patch particulier puis son déploiement sur leur machine afin de faire évoluer la version du jeu. Ces nouvelles versions peuvent potentiellement nécessiter une mise à niveau du matériel côté client (par exemple pour supporter une amélioration du moteur graphique) alors qu'elle n'est pas requise dans le Cloud gaming. Ainsi, la récupération du patch, son installation et la capacité d'utiliser la nouvelle version du jeu peuvent entraîner des coûts supplémentaires.

Ces coûts supplémentaires ne sont généralement pas présents dans l'approche service où le client paie pour la fonction alors que dans l'approche composant le client paie pour l'élément à un instant et dans une version donnée.

Cependant, le principal inconvénient de cette relation service entre client et fournisseur est la dépendance totale du premier face au système du second ainsi que la dépendance aux différents supports de communications entre eux. De fait, une défaillance au niveau de

Tab. 4.1 – CBSE vs SOSE : répartition des responsabilités

	Fournisseur	**Consommateur**
CBSE	Développement, qualité de service, maintenance.	Déploiement, exécution, gestion des fonctionnalités, utilisation
SOSE	Développement, qualité de service, maintenance, déploiement, exécution, gestion des fonctionnalités.	Utilisation

ces éléments qui se trouvent hors du domaine d'action du client voit son incapacité à agir sur la cause. En contre-partie, c'est le contrat préalablement établi avec le fournisseur qui caractérise les conséquences de ces défaillances en terme de compensation pour le client.

Dans le cadre du Cloud gaming, ces défaillances hors contrôle du client sont, par exemple, une erreur au niveau de la plate-forme de jeu ou encore des pertes de connexions internet liées au fournisseur d'accès.

Ainsi, l'orienté service pousse la responsabilité du propriétaire à son extrême par rapport à l'orienté composant et en conséquence réduit la responsabilité du client. En effet, l'approche *off-the-shelf* CBSE implique que le fournisseur est uniquement responsable du développement de son composant, de la qualité de service associée et de la maintenance. Dans l'approche SOSE, le fournisseur est aussi responsable du déploiement de son service, de son exécution et de sa gestion. Le consommateur du service est uniquement responsable de la communication et du respect des contraintes d'utilisation.

Le tableau 4.1 fait le récapitulatif de la répartition des responsabilités entre fournisseurs et consommateurs en CBSE et SOSE.

La vision SOSE de responsabilité accrue du fournisseur implique aussi la nature *multitenant* du service.

4.3.1.3 Nature multitenant du service

Une application est dite *multitenant* [Jac05] si elle offre ses fonctionnalités à de nombreux utilisateurs en parallèle. Elle gère donc de nombreuses exécutions en même temps et doit s'assurer de l'isolation du contexte de chacune d'elles afin de garantir la correction des résultats à ses différents clients.

De la même façon, un service en exécution est voué à gérer de multiples connexions parallèles. Dans notre exemple issu des jeux vidéo, les plate-formes de Cloud gaming supportent un grand nombre de joueurs en parallèle. Pour chacun de ces joueurs, elle doit maintenir un contexte particulier afin de conserver leurs informations respectives. Ces informations sont de deux natures :

- *contrat* : regroupe l'ensemble des données liées au contrat passé entre le client et le fournisseur qui régissent l'utilisation du service (dans notre exemple : abonnement mensuel, numéro de compte, qualité, etc.) ;
- *exécution* : regroupe l'ensemble des données nécessaires à l'exécution de l'application tout au long de l'utilisation du service (dans notre exemple : expérience acquise,

parties jouées, univers persistant, etc, afin de reprendre exactement dans l'état où le joueur s'était arrêté dans sa partie).

Ce principe multitenant n'est pas nécessaire à un composant. En effet, bien que pouvant appartenir à de multiples compositions, au niveau de l'exécution, différentes instances du composant sont créées et chacune d'elles sous la responsabilité d'un client dans le cadre d'une composition particulière.

4.3.2 Différence de rapport au couplage

Le couplage est une notion que nous identifions comme un des points de rupture clé entre CBSE et SOSE. Cette notion exprime toutes les dépendances possibles entre entités conceptuelles ou logicielles. Réduire le couplage garantit un certain nombre de bénéfices intuitifs en termes d'isolation des erreurs, de facilité d'ajouts et de retrait d'entités réutilisés, reconfiguration, etc.

La différence d'importance accordée au couplage faible entre CBSE et SOSE est directement issue de leur différence d'orientation technique. De fait, le CBSE a une vocation généraliste dans le type d'applications qu'il veut permettre d'implémenter, tandis que les mécanismes du SOSE sont construits pour supporter le développement d'applications qui s'exécutent sur des environnements hautement volatiles et hétérogènes. Cette différence s'illustre consécutivement par leur rapport respectif face à la gestion des hétérogénéités d'une part, et à l'automatisation des mécanismes d'autre part.

4.3.2.1 Gestion des hétérogénéités

L'objectif du service est l'indépendance avec les technologies d'implémentation. Un service doit être accessible et utilisable sans aucune hypothèse sur son implémentation, sur les utilisateurs potentiels ou sur la façon d'utiliser ce service. Cette problématique est bien connue en CBSE mais ne représente pas un enjeu aussi critique que dans le SOSE. De fait, il existe un très grand nombre de modèles de composant [CCSV07]. Pour développer un nouveau système, l'architecte doit choisir un modèle particulier et n'utiliser que des composants respectant ce modèle car la collaboration entre modèles différents est très difficile [CCL06]. Ainsi, bien que le CBSE a prouvé son efficacité dans la réutilisation logicielle et la maintenabilité, il ne cible pas spécifiquement certaines difficultés rencontrées par les développeurs liées aux variations de plate-formes, de protocoles, d'appareils, Internet, etc. [BL07]

De son côté, le SOSE prône un unique modèle homogène de services [ES08], devant être standardisé et utilisé par tous, pour encapsuler tous les types de ressources et cacher leur nature hétérogène lors du développement.

4.3.2.2 Rapport à l'automatisation des mécanismes

L'automatisation participe à la définition même du SOSE, et en conséquence, la grande majorité des travaux cherche à automatiser ses mécanismes tels que la publication de services, les découvertes et sélections, la composition, etc. De fait, le découplage entre les

besoins et les services utilisés, leurs découvertes au runtime, la définition des collaborations et enfin l'établissement dynamique des communications ont été des objectifs fixés dès le départ dans l'élaboration du paradigme service. Ce principe d'automatisation est poussé à son extrême par le concept d'auto-adaptation [NGM+08] qui cherche à coordonner l'ensemble des mécanismes liés au service afin d'assurer des adaptations contextuelles réactives voire proactives.

Bien que l'automatisation des processus est un élément clé de la recherche en CBSE et représente bon nombre de ses challenges actuels, elle ne fait pas partie intégrante de l'origine conceptuelle du CBSE ni de la définition d'un modèle de composant [CCSV07].

Ainsi, de par son rapport à l'hétérogénéité et à l'automatisation, le SOSE vise un couplage faible à tous les niveaux, du développement à l'exécution.

4.3.3 Différence de granularité

Dans le domaine de l'ingénierie logicielle, la granularité correspond à une mesure relative de la taille des éléments architecturaux qui construisent les applications. La communauté du génie logiciel parle ensuite de systèmes à gros grains ou à petits grains [BBE+] qui sont associés respectivement à leur construction par assemblage de blocs logiciels à forte granularité ou à faible granularité. Ces notions de forte et faible granularité sont déterminées par l'importance des ressources encapsulées par les éléments architecturaux. Cette importance est relative à la complexité sous-jacente à la réalisation et à l'exploitation de ces ressources.

La maîtrise de la granularité est devenue prépondérante avec le développement du CBSE [MZ08, BBE+]. De fait, cette granularité représente un des premiers points de différenciation entre un objet et un composant. Ce dernier apporte une réponse au manque de lisibilité, de compréhension et donc de manipulabilité de systèmes découpés en un trop grand nombres d'objets ou en objets de trop grande taille. Ainsi, différents modèles de composants proposent différentes granularités [BBE+], et ces variabilités offertes sur la taille de la brique de base ont renforcé l'importance du choix dans le découpage de l'application afin de maximiser la qualité de l'architecture résultat.

La notion de granularité est intuitivement compréhensible ce qui contre-balance avec l'imprécision de sa formalisation où la délimitation claire entre forte granularité et faible granularité reste encore à être définie. Cependant, la compréhension actuelle est suffisante pour établir d'une hiérarchie entre CBSE et SOSE, où les services sont généralement décrits comme des grains plus gros que les composants, de la même façon que les composants sont typiquement vus comme gros grains face aux objets plus petits grains.

Nous justifions ce rapport entre composant et service par deux réalités couramment rencontrées :

– réalité technique : où les modèles de composants sont très souvent utilisés pour construire de nouveaux services SOSE à partir de zéros ou bien à partir de systèmes hérités. Les technologies CBSE associées peuvent intervenir à tous les niveaux de la réalisation d'un système SOSE, de l'implémentation des services, à leurs adaptations pour les intégrer face aux hétérogénéités (environnements d'exécution, langages,

protocoles, interfaces, etc.) ou encore afin de fournir l'abstraction nécessaire à la composition de services préexistants. Ce rapport entre services et composants est le même entre composants et objets où les objets sont couramment utilisés pour implémenter des composants.
- réalité conceptuelle : liée à la nature même du service et des processus qui lui sont associés. Les sections précédentes ont mis en évidence un ensemble de propriétés inhérentes aux SOSE telles que le couplage faible, la gestion des hétérogénéités, le rapport à l'automatisation, les répartitions des responsabilités ou encore le multitenant. Bien que ces notions soient déjà présentes dans le CBSE, l'idée directrice du SOSE est de les pousser à leur extrême. Pour garantir ces avancées, des processus complexes doivent donc s'exécuter. Ainsi, la nature uniquement gros grain du service vient d'un besoin d'adéquation entre le coût du support des processus services, la taille des ressources encapsulées et la pertinence de leur mise sur le réseau.

La réalité technique est à contre-balancer face à des approches composant telles que [OAS09, AAA08] qui, lors de la réalisation d'applications SOSE, voit les services de l'interface d'un composant comme un service au sens SOSE du terme. Dans cette optique, le service SOSE est vu comme une sous-partie de l'interface. Cependant, le rapport de l'implémentation de l'un par l'autre reste le même.

4.3.4 Principales préoccupations de OO, CBSE et SOSE

Dans [CCL06], les auteurs définissent le CBSE par les mots clés *réutilisabilité* et *composabilité*. En nous basant sur les analyses précédentes, nous définissons le SOSE par la *réutilisabilité*, la *composabilité* et la *dynamicité*.
- *Réutilisabilité* : support et facilité d'un produit ou d'un processus lié à un paradigme de développement logiciel à être réutilisé à l'identique ou à travers un certain nombre de modifications.
- *Composabilité* : support et facilité d'un paradigme de développement logiciel à combiner de façon sûre ses éléments architecturaux afin de construire de nouveaux systèmes ou éléments architecturaux composite.
- *Dynamicité* : support et facilité d'un paradigme à développer des applications capables d'adapter de façon dynamique, automatique et autonome leurs comportements pour répondre aux changements d'exigences et de contextes ainsi qu'aux possibilités d'erreurs.

Ces trois critères représentent l'essence qualitative qui a dirigé la définition des paradigmes de développement objet, composant et service. La figure 4.1 illustre cette analyse et offre une vision haut niveau de leurs points d'intérêt primaires et retrace l'évolution chronologique des préoccupations de la communauté du génie logiciel.

La réutilisabilité représente la préoccupation la plus ancienne des trois. Les premiers développeurs se sont rapidement rendus compte des répétitions de code dans une application et ont donc cherché à définir des mécanismes permettant de les limiter. L'OO est centré sur cette préoccupation et son développement est un des aboutissements de

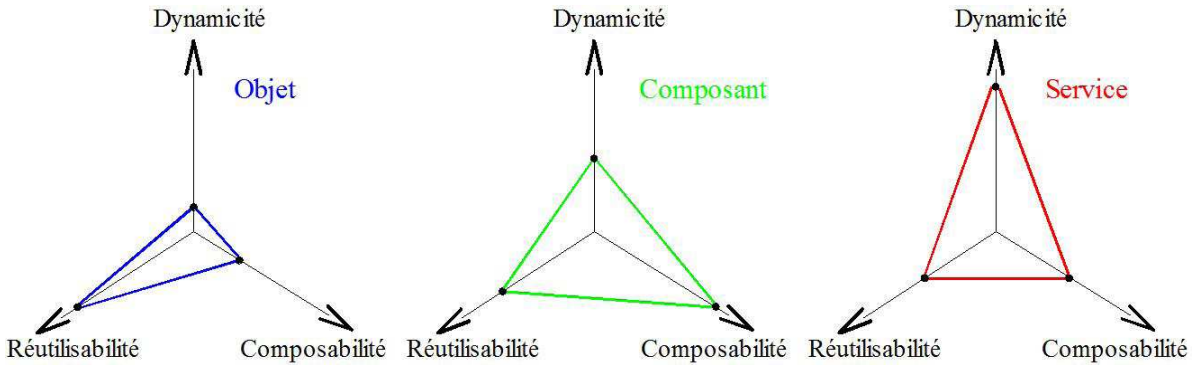

Fig. 4.1 – Évolution des préoccupations globales entre objet, composant et service

cette recherche. La notion d'objet facilite la conservation et la transmission de l'expérience acquise à travers les différents systèmes. Il approfondit la réutilisation, qui se voulait au départ comme une reprise à l'identique de l'existant, par le processus d'héritage qui permet de faire évoluer les données et les comportements sauvegardés afin de les spécialiser aux besoins. Ainsi, l'OO apporte une grande maîtrise de la réutilisabilité qui a ouvert la voix à des applications de plus en plus complexes et donc à l'identification de nouvelles limites en termes de granularité, d'architecture de logiciels, d'abstraction de communication, etc. Ces limites ont donc abouti à un déplacement des préoccupations vers la composabilité.

Ainsi, la communauté de l'ingénierie logicielle a développé et introduit le CBSE pour surmonter ce nouveau challenge. "*Les composants sont pour la composition*", une expression célèbre de Szyperski [Szy02], illustre parfaitement cet état de fait. Par définition, un composant doit avoir un design spécifiquement établi pour supporter les potentialités de composition afin d'assurer l'interopérabilité. Les modèles de composants et les technologies associées (CORBA Component Model CCM [OMG08], COM+ [Mic11], Fractal [BCL+06], etc.) existent pour fournir les cadres précis de développement et de déploiement nécessaires au support des patrons de composition. De tels modèles imposent des formats de composants en termes de construction de codes et de règles de déploiement [CCSV07]. Ainsi, le CBSE renforce la maîtrise de la composabilité et formalise clairement les processus associés. Par extension, cette formalisation pose le socle de base nécessaire aux possibilités d'automatisation. Une partie de la communauté logicielle s'est donc réorientée vers la préoccupation de dynamicité comme l'aspect prédominant.

Ainsi, le SOSE a été développé à partir de l'expérience acquise par les objets et les composants mais en se focalisant dès le départ sur les moyens d'améliorer la dynamicité. Comme présenté dans le chapitre 1, le service est associé à un certain nombre de notions qui servent de support à l'automatisation des mécanismes liés à la réutilisation des ressources et à leur composition pour la définition de nouvelles applications. Le SOSE cherche à apporter une réponse adaptée aux environnements hautement volatiles et ainsi surmonter les limites posées par la vocation généraliste du CBSE.

La figure 4.1 résume ces déplacements de préoccupations. La recherche issue de l'OO se focalise principalement sur la réutilisation et aborde peu la composabilité et la dynamicité.

Le CBSE se focalise sur la composabilité qui participe au renforcement de la réutilisabilité mais cherche aussi à automatiser ses processus. Le SOSE se focalise principalement sur la dynamicité des processus existants qui assurent réutilisation et composition.

La comparaison directe suivant ces trois critères de qualité entre objet, composant et service (qui est le plus réutilisable, le plus composable, le plus dynamique) est très difficile à établir car elle dépend du point de vue de celui qui compare. Les résultats varient en fonction des contextes dans lesquels il se positionne et il positionne les entités logicielles objet, composant et service. Par exemple, d'un point de vue développeur bas niveau un objet sera plus facilement réutilisable qu'un service alors qu'inversement, d'un point de vue métier un service très haut niveau sera plus facilement réutilisable.

Ainsi, notre cadre conceptuel de comparaison tente de prendre en compte cette réalité en fournissant à ces utilisateurs l'ensemble des éléments permettant d'exprimer leurs propres analyses et comparaisons qualitatives. Ces aspects qualitatifs se basent sur une classification des supports fournis par les paradigmes que nous avons regroupés dans les aspects quantitatifs.

4.4 Aspects quantitatifs

Cette section classifie les éléments structurels et les mécanismes qui caractérisent le CBSE et le SOSE. Nous ne cherchons pas à être exhaustifs mais plutôt à mettre en évidence leurs différences par une liste pertinente des concepts centraux. Ces concepts sont répertoriés dans les catégories produits et processus.

4.4.1 Produits et processus

Un **produit** est une entité logicielle ou conceptuelle qui est le résultat d'une action ou d'un processus. Un **processus** est une action ou une succession d'actions qui est utilisée pour créer ou modifier un produit et ainsi obtenir un produit en résultat. Les produits sont répartis en deux sous catégories :

- **éléments architecturaux simples** : les briques élémentaires de construction d'un paradigme ;
- **éléments architecturaux composites** : les produits complexes construits à partir d'éléments architecturaux existants. Leur structure identifie clairement les éléments architecturaux réutilisés et leurs relations.

Chaque sous catégorie est à son tour divisée en deux groupes suivant les deux *niveaux d'abstraction* : le *design-time* et le *runtime*.

La catégorie des processus se focalise sur le principe de réutilisation, c'est-à-dire comment réutiliser des entités logicielles, les *constituants*, pour en construire de nouvelles, les *composites*. De manière classique, un constituant peut être un élément architectural simple ou composite. Ces notions de constituants et composites définissent les deux *niveaux de description*. Ainsi les processus sont regroupés suivant les niveaux d'abstraction et de description :

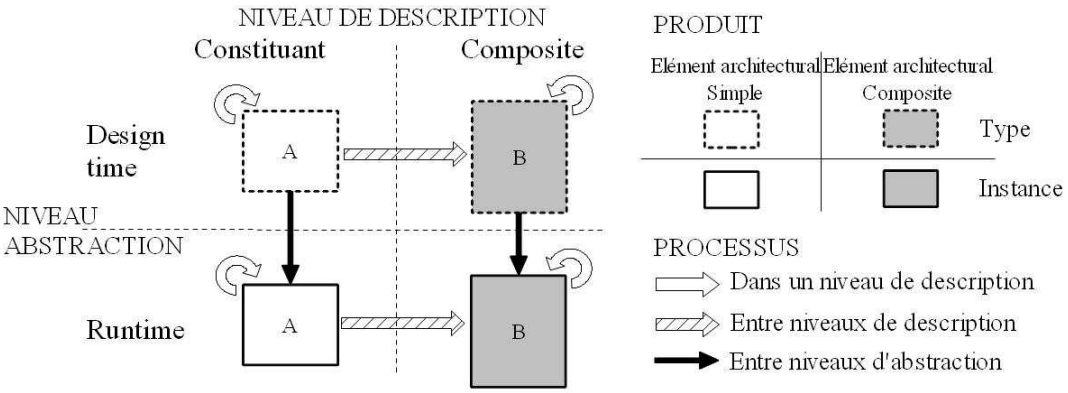

Fig. 4.2 – Niveaux d'abstraction et de description : répartition des produits et des processus

- **dans un même niveau de description** - regroupe les processus qui ciblent et produisent des produits d'un même niveau de description (Figure 4.2 flèches blanches). Cette catégorie est à la fois divisée en design-time et runtime ;
- **entre niveaux de description** - regroupe les processus qui ciblent des produits de deux niveaux de description différents (Figure 4.2 flèches hachurées). Cette catégorie est divisée en design-time et runtime ;
- **entre niveaux d'abstraction** - regroupe les processus qui assurent le passage des produits entre design-time et runtime (Figure 4.2 flèches noires).

La figure 4.2 montre la répartition des produits et processus sur une représentation simple. Un composite B est construit à partir d'un ensemble de constituants tels que A qui est défini comme un élément architectural simple. Ces deux produits ont leurs représentations au design-time et au runtime. Les différentes flèches représentent les processus qui sont étudiés. Les flèches blanches sont les processus liés à un même niveau de description et d'abstraction. Les flèches hachurées sont les processus qui font le lien entre niveaux de description et qui ont leurs représentations au design-time et au runtime. Les flèches noires sont les processus qui font le lien entre niveaux d'abstraction et assurent donc le passage du design-time au runtime.

4.4.2 Comparaison entre objet, composant et service

4.4.2.1 Produit

Éléments architecturaux simples - (Tableau 4.2) Les éléments architecturaux simples de l'orienté objet sont la *classe* au design-time et son instance, l'*objet*, au runtime. La même distinction est faite pour le CBSE, entre les produits *type de composant* et *type de connecteur* [AO09, GMW97], et leurs instances *composant* et *connecteur*.

Les connecteurs [CCSV07] sont des médiateurs dans les connexions entre composants, c'est-à-dire qu'ils se posent en intermédiaires entre ces composants constituants. Ils ont un double emploi : (i) de permettre les compositions indirectes ou exogènes entre

composants, par opposition aux compositions directes ou endogènes, et (ii) d'introduire des fonctionnalités additionnelles au travers du code glue qu'ils encapsulent.

En SOSE, la frontière entre les niveaux d'abstraction est beaucoup moins claire et la plupart des travaux existants se réfèrent à un *service* comme une entité du runtime [SD05, The08]. Néanmoins, une notion de service abstrait existe dans certaines approches [CNP09]. Cette notion sert à faire la distinction entre l'expression des besoins recherchés par l'architecte pour définir son application et les services concrètement disponibles dans le système pour répondre à ces besoins. Cependant, une clarification exacte entre service abstrait et service concret reste encore à définir. Nous citons également la notion de *description de service* qui est un produit capital du SOSE [OAS08]. De fait, chaque représentation runtime d'un service est associée à sa *description de service* (chapitre 1.2.3.1) qui est la cible de nombreux processus impliqués dans l'exploitation des ressources.

Élements architecturaux composites - (Tableau 4.2) Les trois paradigmes partagent la notion de *composite*. L'orienté objet repose sur les notions de *classe composite* et d'*objet composite*. Le CBSE repose au design-time sur les notions de *type de configuration* et *type de composant composite*, et au runtime sur leurs instances *configurations* et *composants composites*.

La notion de composition de services et par extension de service composite du SOSE est principalement au runtime. En effet, la plupart des travaux existants considèrent le service composite comme l'exécution par un moteur de composition d'un schéma de collaboration entre services. Cependant, certaines approches [GMJ08, ZBD+03] introduisent des notions d'instanciation de schéma de collaboration à partir de "*templates*" abstrait qui les décrivent. Nous choisissons de prendre en compte cette représentation similaire à l'OO avec des *types de schéma de collaboration*, comme entités du design-time, et des *instances de schéma de collaboration*, comme entités du runtime. De plus, un schéma de collaboration est classiquement associé à deux patrons de coordination de services, la *chorégraphie* et l'*orchestration* [OAS08] qui ont des technologies supportant leur représentation au design-time et au runtime. Cette approche type et instance de schéma de collaboration est aussi celle de nous avons appliqué dans notre méta-modèle de service composite (section 2.2.3.1).

Le service composite encapsulant une composition de services, nous le définissons d'une façon similaire en *type de service composite* et *instance de service composite*.

4.4.2.2 Processus

L'objectif de cette partie n'est pas de fournir une liste exhaustive des processus existants mais de sélectionner les plus pertinents et les plus largement acceptés dans la communauté afin de mettre en perspective les différences entre objet, composant et service.

Dans un même niveau de description - (Tableau 4.2)

– **Design-time** : l'orienté objet repose principalement sur les processus d'*association* et d'*héritage*. Le CBSE repose sur la *composition horizontale* [Bar06, CCSV07] entre éléments architecturaux de même niveau de description. Cette composition

Tab. 4.2 – Produit-Processus : comparaison entre objet, composant et service

Produit		**OBJET**	**COMPOSANT**	**SERVICE**
Élément architectural simple	**Design-time**	Classe	Type composant, Type connecteur	Service abstrait
	Runtime	Objet	Composant, Connecteur	Service concret, Description de Service
Élément architectural composite	**Design-time**	Classe composite	Type de configuration, Type composant composite	Type de Schéma de collaboration, Type de Service composite
	Runtime	Objet composite	Configuration, Composant composite	Instance de Schéma de collaboration, Instance de Service composite
Processus				
Dans un niveau d'abstraction	**Design-time**	Association, Héritage	Composition horizontale, Héritage sélectif, Versionnement, Raffinement	Chorégraphie,
	Runtime	Appel de méthode	Appel de fonction	Chorégraphie, Découverte et sélection, Invocation, Publication
Entre niveaux de description	**Design-time**	Composition	Composition verticale	Orchestration,
	Runtime	Appel de méthode	Appel de fonction, Délégation	Orchestration, Invocation, Découverte et sélection
Entre niveaux abstraction		Instanciation	Instanciation	Découverte et sélection, Instanciation de schéma

horizontale correspond au processus d'établissement des connexions entre composants constituants.
On peut également citer les processus de *versionnement*, d'*héritage sélectif* et de *raffinement*.
Les processus SOSE dans un même niveau de description se focalisent principalement sur la manipulation des schémas de collaboration entre services constituants. Nous citons le processus de *chorégraphie* qui est un des supports centraux de la réutilisation et qui exprime les communications directes entre constituants.

– **Runtime** : les processus de communication entre éléments architecturaux sont la préoccupation majeure de cette catégorie. OO et CBSE reposent principalement sur différents processus d'*appels* de fonctionnalités, alors que le SOSE inclut obligatoirement des processus additionnels. Typiquement, les services constituants doivent être découverts et sélectionnés dynamiquement (processus de *découverte et de sélection de services*). Ensuite ces services constituants coordonnent leurs actions par un processus de *chorégraphie* qui définit les successions d'*invocations de service*. De plus, un processus préalable de *publication de services* est nécessaire pour mettre un service à disposition des clients potentiels.

Entre niveaux de description - (Tableau 4.2)

– **Design-time** : l'OO repose sur le processus de *composition* pour produire des éléments composites. Le CBSE repose sur la *composition verticale* qui fait le lien entre constituants et composites. La composition verticale [Bar06, CCSV07] (ou composition hiérarchique) consiste à avoir un sous-composant (le constituant) encapsulé dans un composant composite. Cette composition est irréflexive afin d'éviter les cycles, c'est-à-dire qu'un même composant ne peut pas se retrouver à plusieurs niveaux de la hiérarchie. Elle supporte les combinaisons consistantes des comportements du composite avec les comportements des constituants. De plus, les constituants sont cachés face aux requêtes des clients du composite.
Le SOSE repose sur le processus d'*orchestration* qui modélise les communications verticales entre le service composite et ses services constituants.
– **Runtime** : les processus de communication entre constituants et composites sont l'essence de cette catégorie. OO et CBSE reposent sur différents processus d'*appels*. En CBSE, ces appels sont parfois nommés comme le processus de *délégation*.
En SOSE, la coordination des processus d'*invocations de services* entre du composite vers ses constituants est appelée l'*orchestration*. De la même façon, les processus de découvertes et sélection dynamique de services sont nécessaires au service composite pour identifier ses constituants.

Entre niveaux d'abstraction - (Tableau 4.2) Objet et composant reposent sur le processus d'*instanciation* pour lier type et instance.

Pour le SOSE, nous avons proposé l'emploi des services abstraits et services concrets comme respectivement des éléments du design-time et runtime. Cependant, le passage de l'un à l'autre repose sur les processus de *découverte et sélection de services*. Cette particularité est un élément de différenciation majeure avec les orientations objet et

composant. D'une façon similaire à [GMJ08, ZBD+03], le passage d'un type de schéma de collaboration à une instance de schéma de collaboration repose sur l'*instanciation*. Le passage d'un type de service composite à une instance de service composite correspond à la combinaison des découvertes et sélections de ses services constituants et de l'instanciation des schémas de collaboration qui dirigent son comportement.

4.5 Aspects qualitatifs

Les travaux existants autour de la qualité logicielle définissent un grand nombre de critères tels que la performance, la robustesse, la flexibilité, la maintenabilité, etc [KL96, BKM07]. Chacun de ces travaux fournit leur propre organisation de ces critères et développent des méthodes de mesures dédiées. La définition de ces critères de qualité et la façon de les appréhender sont adaptées au point de vue de l'utilisateur ciblé par ces cadres de mesures. En effet, la compréhension d'une qualité peut varier entre les acteurs intervenant, qu'ils soient architectes, développeurs, designers ou autre. De plus, le domaine d'application du système influence directement l'importance accordée à ces critères. Certaines applications favorisent les performances, alors que d'autres se focalisent sur la sécurité des exécutions.

Nous cherchons donc à couvrir l'ensemble de ces variabilités en offrant la possibilité aux utilisateurs de définir leur propre vision des qualités qui les intéressent. Dans une premier temps, nous identifions l'ensemble des propriétés des paradigmes de développement qui influencent la qualité logicielle. Puis, nous comparons les approches objet, composant et service suivant ces propriétés. Dans un second temps, l'utilisateur définit les critères de qualité qu'il veut mesurer en combinant les résultats précédents. Il spécifie l'importance respective des propriétés sur la qualité ciblée et quelles sont leurs influences mutuelles dans ce contexte particulier.

4.5.1 Propriétés qualitatives de comparaison des paradigmes de développement

Nous avons identifié six catégories de propriétés qualitatives principales qui sont communes à tout paradigme de développement logiciel. Ces catégories regroupent les éléments qui ont des impacts significatifs sur la qualité globale, du processus de développement au système produit en résultat.

Couplage faible : potentiel de réduction des dépendances entre produits et des dépendances entre processus.

Architecture explicite : capacité d'un paradigme à définir des vues architecturales claires d'une application, c'est-à-dire fournir les moyens d'identifier et d'expliciter les fonctionnalités associées aux produits qui composent l'application ainsi que les processus entre ces produits.

Abstraction de communications : capacité d'un paradigme à abstraire les communications des autres fonctions de l'application puis à appréhender et comprendre

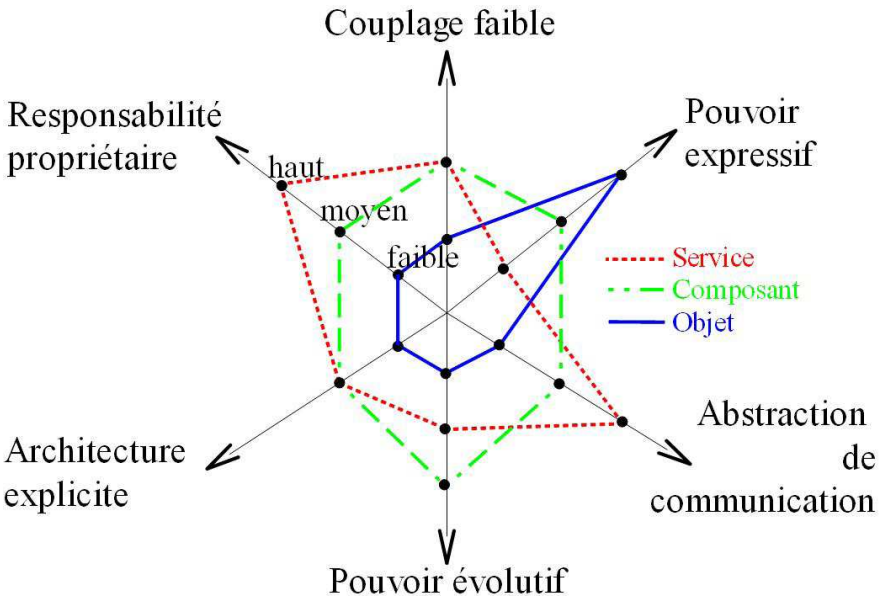

Fig. 4.3 – Comparaison des propriétés entre objet, composant et service

ces communications d'un seul tenant afin de pouvoir facilement les manipuler. Cette séparation des préoccupations est accrue par un découplage entre l'identification des communications par le design de l'architecture et la réalisation physique de ces communications. Cela représente la capacité de s'abstraire des problématiques physiques d'hétérogénéités des plate-formes, du déploiement et de l'exécution, etc.

Pouvoir expressif : potentiel d'expression du paradigme en terme de capacité et d'optionalités de création. Il se base sur le nombre de concepts et de processus fournis pour spécifier, développer, manipuler, exécuter et implémenter des applications. Cette catégorie n'établit pas de jugement de valeur entre les différents concepts mais répertorie uniquement leur variété.

Pouvoir évolutif : potentiel d'un paradigme à faire évoluer ses produits et processus. Il se base sur une analyse et un jugement de valeur portés sur les différents processus qui supportent ces évolutions et sur leurs cibles.

Responsabilité propriétaire : correspond à la répartition des responsabilités entre fournisseurs et consommateurs. Ces responsabilités se focalisent sur l'entité logicielle réutilisée en terme de : développement, qualité de service, maintenance, déploiement, exécution, gestion. Cette répartition exprime le degré de liberté accordé aux consommateurs par le fournisseur.

4.5.2 Comparaison Objet, Composant et Service

La figure 4.3 montre un exemple d'utilisation des six propriétés pour évaluer les différences entre OO, CBSE et SOSE. Les résultats sont donnés suivant trois niveaux d'importance (haut, moyen, faible) accordés sur chacune des propriétés et expriment notre

analyse de l'état actuel des trois paradigmes de développement. Cette comparaison établit une évaluation relative entre les trois paradigmes (l'un supérieur à l'autre).

Couplage Faible - typiquement, les systèmes à base d'objets font intervenir un ensemble de classes fortement couplées tandis que CBSE et SOSE ciblent une réduction de ce couplage. Le couplage faible est un enjeu clé de ces deux paradigmes. L'entrelacement des solutions proposées et leurs influences mutuelles ne permettent pas de les différencier de manière claire. Cependant, notre étude du couplage, présenté dans le chapitre 3, permet d'identifier une marge de progression possible.

Architecture explicite - contrairement à l'OO, CBSE et SOSE reposent directement sur le concept d'architecture logicielle.

Abstraction de communication - le SOSE fournit la meilleure abstraction de communication, basée sur l'encapsulation apportée par les services et l'isolation des communications dans un schéma de collaboration. Le comportement global est plus facilement localisable, compréhensible et manipulable. De plus, cet aspect est renforcé par la nature multitenant du service et sa capacité à gérer des exécutions parallèles. En CBSE, les communications sont localisées à l'intérieur des différents connecteurs qui partagent le comportement global. Les flots de travail sont moins explicites. La granularité très fine du OO accentue cet handicap d'explosion des collaborations entre les différentes associations et appels de méthodes entre classes.

Pouvoir expressif - l'OO manipule un grand nombre de concepts tels que l'héritage, les niveaux d'abstraction, les niveaux de description, la granularité, la réflexion, etc. Ces concepts s'expriment au travers de différents langages de programmation puissants dans leur capacité à créer des systèmes et des nouvelles fonctionnalités. Le CBSE s'est largement inspiré de ce travail mais il n'a pas encore atteint le même niveau de maturité sur des concepts tels que l'héritage, la réflexion, etc. De plus, il n'existe pas de langages de programmation composant équivalents aux langages objet du point de vue de leur capacité à créer des fonctionnalités, mais un ensemble de langages de description des assemblages de composants liés aux différents modèles existants. Enfin, le SOSE possède le plus faible pouvoir expressif car il combine les même lacunes que le composant auxquelles s'ajoutent des imprécisions sur les niveaux d'abstraction soulignés par notre analyse quantitative. De plus, il n'existe pas de "*langage spécifique service*", à la manière de langage de description d'architecture à composants, qui permette à lui seul de décrire et construire des applications SOSE.

Pouvoir évolutif - le pouvoir évolutif est directement lié à la notion d'architecture explicite. Une architecture logicielle définit clairement des entités et les relations entre ces entités. On peut la représenter par un graphe basé sur des noeuds et des arcs. Les processus d'évolutions peuvent être regroupés suivant leur cible, le noeud, l'arc ou le graphe. Typiquement, l'OO ne propose pas cette notion d'architecture explicite et donc de graphe. Les processus d'évolution de l'OO se focalisent uniquement sur les noeuds et les arcs. Au contraire, CBSE et SOSE manipulent une architecture explicite et donc offrent des processus d'évolutions sur les trois cibles. Cependant, la maturité plus importante du CBSE et sa gestion explicite des niveaux d'abstraction ont permis à sa communauté d'aller encore plus loin et de proposer des processus d'évolutions aux niveaux meta et

meta meta architecture [GTOS08, GBSC09].

Responsabilité propriétaire - le SOSE pousse la responsabilité propriétaire à son extrême où le fournisseur de service est responsable du développement, de la qualité de service, de la maintenance, du déploiement, de l'exécution et de la gestion. Au contraire, le CBSE partage les responsabilités au niveau du déploiement où le client devient responsable de l'instanciation du composant dans son application, de son exécution et de sa gestion. En OO, la classe est typiquement en boîte blanche où le client est libre de la manipuler à sa guise mais il en possède l'entière responsabilité.

Cette comparaison en six propriétés des paradigmes objet, composant et service représente une instance particulière suivant notre propre expertise dans le domaine. La méthode d'analyse utilisée permet uniquement d'établir des mesures de hiérarchies, c'est-à-dire qu'elle statue un ordre relatif entre les paradigmes comparés où l'un est plus efficace que l'autre sur une propriété particulière. Ainsi, ce cadre de comparaison peut être réutilisé pour évaluer d'autres paradigmes ou des technologies implémentant ces paradigmes (voir annexe A). Cependant, les résultats obtenus sont, dans l'état actuel du cadre, limités à des hiérarchies relatives.

L'établissement de cette comparaison entre OO, CBSE et SOSE est la première étape à leur évaluation qualitative respective.

4.5.3 Perspectives d'analyse qualitative

Dans ce travail, nous ne cherchons pas à fixer une n-ième liste de définitions de critères de qualité, mais nous voulons plutôt offrir les moyens d'évaluer ces qualités quelque soit la manière dont l'utilisateur les appréhende. De fait, la vision qu'a un utilisateur sur ces critères de qualités logicielles dépend de son expérience personnelle dans le domaine. Par exemple, le principe de flexibilité d'une application représente une notion immédiatement compréhensible pour tout acteur impliqué (architecte, designer, développeur, vendeur etc.), cependant sa formalisation précise varie d'un acteur à un autre, et même d'un spécialiste à l'autre.

Ainsi, notre cadre de comparaison conceptuel est construit de manière à laisser la place à la définition par l'utilisateur de ses propres *perspectives qualitatives d'évaluation*. L'approche choisie est que ce dernier exprime ses connaissances en spécifiant la perspective à travers laquelle il veut étudier les trois paradigmes de développement comparés. Une perspective particulière correspond au focus de cet utilisateur sur une qualité précise. Il définit une formule d'évaluation de cette qualité en combinant les résultats issus de la comparaison précédente (section 4.5.2), c'est-à-dire suivant les mesures de nos six propriétés : couplage faible, pouvoir expressif, abstraction de communication, architecture explicite, pouvoir évolutif et responsabilité propriétaire.

4.5.3.1 Définition et modélisation d'une perspective qualitative

Une perspective qualitative est la combinaison de i) la qualité logicielle choisie pour comparer les paradigmes et de ii) l'expression de l'expertise de l'utilisateur par rapport

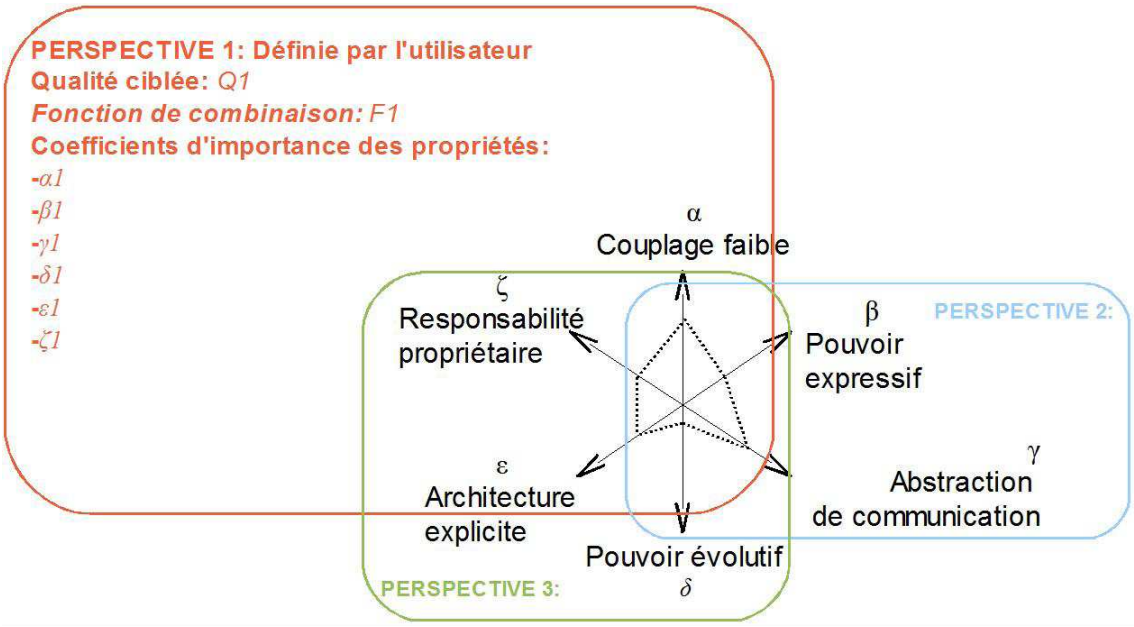

Fig. 4.4 – Modélisation d'une perspective qualitative

à cette qualité. Ainsi, nous définissons une formule type qui modélise cette capacité de personnalisation :

$$Quality = F(\alpha, \beta, \gamma, \delta, \epsilon, \zeta) \tag{4.1}$$

Les six différents coefficients de α à ζ expriment l'importance respective des six propriétés de comparaisons (Figure 4.4) qui est accordée par l'utilisateur en rapport avec la qualité cible. La fonction F définit la manière dont sont combinés les coefficients et les mesures des propriétés.

Une *perspective* est donc une *fenêtre qualitative* posée sur les six propriétés et leurs résultats. Pour illustrer une utilisation, nous nous plaçons comme un utilisateur quelconque. Nous évaluons les trois paradigmes précédemment mesurés, objet, composant et service, suivant notre perspective personnelle sur trois critères de qualité choisis : la *réutilisabilité*, la *composabilité* et la *dynamicité* qui représentent le noyau des préoccupations des paradigmes OO, CBSE et SOSE (section 4.3.4).

4.5.3.2 Exemple de perspectives qualitatives : réutilisabilité, composabilité, dynamicité

La figure 4.5 illustre notre expertise des trois critères de qualité suivant les six propriétés de comparaison des paradigmes. Nous répartissons ces propriétés en trois groupes, en fonction de l'impact qu'elles ont sur les différents critères de qualité, du groupe α qui réunit les propriétés ayant le plus fort impact, au groupe γ qui réunit celles ayant le plus faible. Par exemple, la réutilisabilité est principalement influencée par les pouvoirs

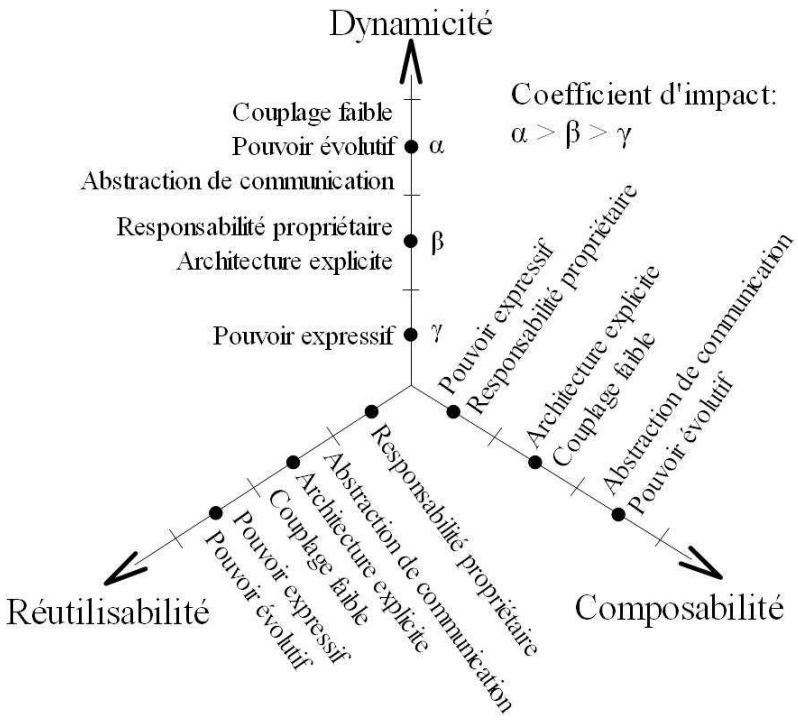

Fig. 4.5 – Expression des perspectives de réutilisabilité, composabilité et dynamicité

expressif et évolutif, puis par l'abstraction de communication, l'architecture explicite et le couplage faible, et enfin par la responsabilité propriétaire (figure 4.5). À partir de là, nous définissons un ensemble de formule qui combine cette répartition et les résultats de la classification précédente des paradigmes de développement (figure 4.3). Pour calculer une mesure numéraire, nous associons un poids à chaque niveau de la figure 4.3 avec respectivement 1, 2 et 3 pour les niveaux faible, moyen et haut. Par exemple pour la réutilisabilité de l'objet, on obtient α multiplié par la somme du pouvoir expressif (haut :3) et du pouvoir évolutif (faible :1) additionné à β multiplié par la somme de l'abstraction de communication (faible :1), de l'architecture explicite (faible :1) et du couplage faible (faible :1), additionné à nouveau à γ multiplié par la responsabilité propriétaire (faible :1).

$$\begin{aligned}
\textbf{Réutilisabilité} =& \mathit{Objet} : \alpha(4) + \beta(3) + \gamma(1) \\
& \mathit{Composant} : \alpha(5) + \beta(6) + \gamma(2) \\
& \mathit{Service} : \alpha(3) + \beta(7) + \gamma(3) \\
\textbf{Composabilité} =& \mathit{Objet} : \alpha(2) + \beta(2) + \gamma(4) \\
& \mathit{Composant} : \alpha(5) + \beta(4) + \gamma(4) \\
& \mathit{Service} : \alpha(5) + \beta(4) + \gamma(4) \\
\textbf{Dynamicité} =& \mathit{Objet} : \alpha(3) + \beta(2) + \gamma(3) \\
& \mathit{Composant} : \alpha(7) + \beta(4) + \gamma(2) \\
& \mathit{Service} : \alpha(7) + \beta(5) + \gamma(1)
\end{aligned} \tag{4.2}$$

Ainsi, on peut remarquer que le SOSE et le CBSE ont des niveaux de composabilité similaires. Cependant, le SOSE possède une meilleure dynamicité tandis que le CBSE a une meilleure réutilisabilité. Ce résultat est directement dépendant de la façon dont nous avons exprimé notre vision des différents critères (Figure 4.5). Un autre utilisateur peut exprimer les critères réutilisabilité, composabilité et dynamicité de façon différente et donc obtenir d'autres résultats.

En résumé, le cadre conceptuel dresse un état des lieux comparatif entre objet, composant et service. Ses différentes catégories identifient les caractéristiques importantes d'un paradigme de développement logiciel et offrent un référentiel commun, applicable pour évaluer l'OO, le CBSE et le SOSE sans faire de favoritisme. Cette évaluation est de nature quantitative et qualitative et permet une compréhension globale de leurs similarités et différences. Cependant, l'évaluation qualitative présentée (section 4.5.2) est uniquement relative, c'est-à-dire qu'elle établit un rapport de supériorité entre paradigme mais ne mesure ni leurs valeurs ni leurs écarts. L'exemple de perspectives, présenté section 4.5.3.2, qui découle de cette évaluation relative fournit donc des résultats à leur tour relatifs.

Dans la suite de ce chapitre, nous discutons des limites de cette évaluation qualitative et expliquons en quoi notre travail sur le couplage faible permet de l'améliorer et représente un premier pas vers une comparaison de mesures absolues.

En parallèle, nous proposons de réutiliser le cadre conceptuel de comparaison pour déterminer les éléments du paradigme SOSE que notre méta-modèle de service composite (chapitre 2) essaie de renforcer à travers ses contributions sur le processus de composition dynamique de services.

4.6 Retour aux contributions à SOSE à travers le cadre de comparaison

Dans un premier temps, cette section se focalise sur notre redéfinition du couplage faible et illustre en quoi elle renforce l'évaluation qualitative du cadre conceptuel de comparaison. Puis, elle dresse un ensemble de perspectives futures d'amélioration de ce cadre dans la continuité du travail sur le couplage. Dans un second temps, cette section replace nos travaux sur le méta-modèle de service composite dans le cadre de comparaison pour identifier les aspects du paradigme service qu'il recherche à améliorer. Cette projection nous permet de déterminer le travail qu'il reste à accomplir dans le SOSE autour de notre vision d'un service composite.

4.6.1 Amélioration de l'évaluation qualitative : l'exemple du couplage faible

La notion de couplage faible est une des six propriétés (section 4.5.1) que nous avons identifiées pour caractériser les perspectives qualitatives et permettre l'évaluation des

paradigmes OO, CBSE et SOSE. Notre travail précédent sur une nouvelle proposition de définition du couplage a donc un impact significatif sur les mesures effectuées par le cadre conceptuel de comparaison.

4.6.1.1 Redéfinition du couplage faible

Dans le chapitre 3, nous avons proposé une nouvelle vision du couplage faible qui repose sur une décomposition en trois couplages : sémantique, syntaxique et physique. Chacun de ces couplages est associé à des métriques d'évaluation particulières qui sont ensuite combinées dans une formule d'évaluation globale. La formule globale permet de mesurer le couplage de compositions de services spécifiques, puis, elle sert de base à la définition d'un cadre de comparaison d'approches de composition.

Ainsi, cette nouvelle proposition offre une meilleure compréhension des implications liées à l'évaluation du couplage et à sa réduction. Elle permet de raffiner la mesure de l'axe de propriété *couplage faible* de la partie qualitative de notre cadre de comparaison (section 4.5.1). De fait, les nouveaux critères de comparaison d'approches de composition de services et la capacité à déterminer le couplage d'une composition cible permettent de hiérarchiser de façon plus fine les différentes propositions existantes. L'état des lieux de la recherche dans le SOSE sur la réduction du couplage est donc plus précis.

Cependant, pour permettre sa comparaison avec les approches objet ou composant, il est nécessaire d'étudier l'applicabilité de notre définition du couplage faible dans les autres paradigmes. Bien que son application semble intuitive, il est nécessaire d'effectuer une recherche plus poussée pour déterminer les possibles biais laissés par sa définition orientée vers le SOSE.

4.6.1.2 Perspectives d'amélioration du cadre conceptuel

L'ensemble des aspects qualitatifs de notre cadre conceptuel de comparaison repose sur les six propriétés précédemment identifiées et sur leur mesure. L'évaluation présentée dans la section 4.5.2 est une comparaison relative entre les trois paradigmes de développement. En effet, elle établit exclusivement une hiérarchie entre ces paradigmes, définissant un rapport de supériorité sémantique[2] de l'un à l'autre. Cette organisation est suffisante pour remplir l'objectif de comparaison directe entre objet, composant et service qui a été fixé, cependant, elle ne permet pas une mesure absolue sur chacune des propriétés. En conséquence, les perspectives d'évaluation qualitative (section 4.5.3) qui se basent sur ces mesures relatives établissent ce même rapport hiérarchique dans leurs résultats.

L'exemple précédent (section 4.5.3.2) qui spécifie les perspectives qualitatives réutilisabilité, composabilité et dynamicité illustre cette limitation où l'analyse de leurs résultats permet uniquement de déterminer une hiérarchie entre OO, CBSE et SOSE suivant ces trois critères.

[2]un paradigme propose plus ou moins de produits ou de processus qu'un autre paradigme pour gérer explicitement une propriété qualitative ciblée

Pour obtenir une évaluation qualitative plus précise, il est nécessaire de raffiner l'analyse des six propriétés et leurs méthodes de mesure de façon à obtenir des résultats absolus qui statuent objectivement la valeur d'un paradigme. De plus, ces nouvelles méthodes devront être aussi applicables sur les technologies associées aux paradigmes. Le but de ce futur travail de raffinement sera de faire le lien entre les concepts haut niveau des paradigmes de développement, que nous avons explicité dans ce chapitre, et les technologies bas niveau qui permettent leur réalisation. L'établissement de ce lien représente la finalité de notre approche top-down qui sera ainsi capable d'offrir une comparaison complète de la théorie à la pratique.

Ainsi, en offrant des mesures plus exactes des six propriétés, les perspectives définies par les utilisateurs seront mécaniquement plus précises et l'analyse des résultats plus riche.

Ce travail a déjà débuté par notre redéfinition du couplage faible présentée dans le chapitre 3 et doit être étendu aux cinq propriétés restantes : *pouvoir expressif*, *abstraction de communication*, *pouvoir évolutif*, *architecture explicite*, *responsabilité propriétaire*.

4.6.2 Rapport du méta-modèle de service composite face au SOSE

Le méta-modèle de service composite, présenté dans le chapitre 2, a pour objectif principal la réduction du couplage dans une composition de services et ainsi, il cherche à améliorer l'axe de propriété *couplage faible* de l'évaluation précédente du SOSE (section 4.3). Cependant, il repose sur un ensemble d'éléments qui ont des implications sur différentes autres caractéristiques du paradigme service. Nous réutilisons les aspects quantitatifs et qualitatifs du cadre conceptuel de comparaison pour mettre en évidence ces implications.

4.6.2.1 Aspects quantitatifs

Le processus de composition de services définit les méthodes et moyens nécessaires pour réutiliser des ressources existantes exposées en tant que services. Notre méta-modèle de service composite, qui réifie et explicite ces méthodes et moyens, se base sur une définition précise du service en *service abstrait* et *service concret*. Les notions de service abstrait et service concret permettent un découpage explicite suivant les niveaux d'abstraction où le service abstrait est défini comme un produit du design-time, et le service concret est défini comme la représentation du service abstrait dans le runtime. Seul le service concret est la cible des processus d'*invocation*. Tous deux sont des éléments architecturaux simples. Enfin, les processus de *découverte et sélection de services* assurent le passage de l'un à l'autre.

D'autre part, le méta-modèle de service composite s'appuie sur différents services gestionnaires dont le gestionnaire de collaboration. Ce gestionnaire de collaboration repose sur les concepts de *type de schéma de collaboration* et d'*instance de schéma de collaboration*. Le type de schéma de collaboration est un produit du design-time. Il représente une composition de services abstraits. Il est obtenu uniquement par un

Tab. 4.3 – Produits et processus du méta-modèle de service composite

			Méta-modèle de service composite
ASPECT QUANTITATIF			
Produit	**Élément architectural simple**	**Design-time**	Service abstrait
		Runtime	Service concret
	Élément architectural composite	**Design-time**	Type de schéma de collaboration, ***Type de service composite***
		Runtime	Instance de schéma de collaboration, ***Instance de service composite***
Processus	**Dans un niveau de description**	**Design-time**	***Héritage de schéma de collaboration***
		Runtime	
	Entre niveaux de description	**Design-time**	Orchestration de services abstraits
		Runtime	Orchestration de services concrets, Invocation de services concrets, ***Auto-composition***
	Entre niveaux abstraction		Découverte et sélection de services, Instanciation de schéma de collaboration, ***Instanciation de service composite***

processus d'*orchestration* entre les services abstraits car notre service composite interdit les communications directes entre services de même niveau de description (interdiction des communications horizontales entre constituants). L'instance de schéma de collaboration est la représentation du type de schéma de collaboration dans le runtime. Le processus qui assure le passage de l'un à l'autre est le processus d'*instanciation*. Ce processus d'instanciation inclut les processus de découverte et sélection sur les services abstraits du type de schéma de collaboration. Cette approche par type et instance s'inspire directement de l'objet. Comme expliqué dans le chapitre 2, elle facilite la réutilisation et peut permettre le développement d'un processus d'*héritage entre types de schéma de collaboration*. Cet héritage sera un support important du multitenant face à l'implémentation de comportements particuliers du service composite pour certains clients qui ont négocié des exécutions personnalisées avec son fournisseur. C'est un processus du design-time dans un même niveau de description.

La première contribution du méta-modèle est d'expliciter la notion de service composite. Il réifie à travers les différents gestionnaires et leurs interdépendances les mécanismes nécessaires qui interviennent dans une composition de services. Dans le prolongement de l'instanciation des schémas de composition du gestionnaire de collaboration, notre service composite est différencié en *type de service composite* et *instance de service composite*. Type et instance sont liés par le processus classique d'*instanciation*.

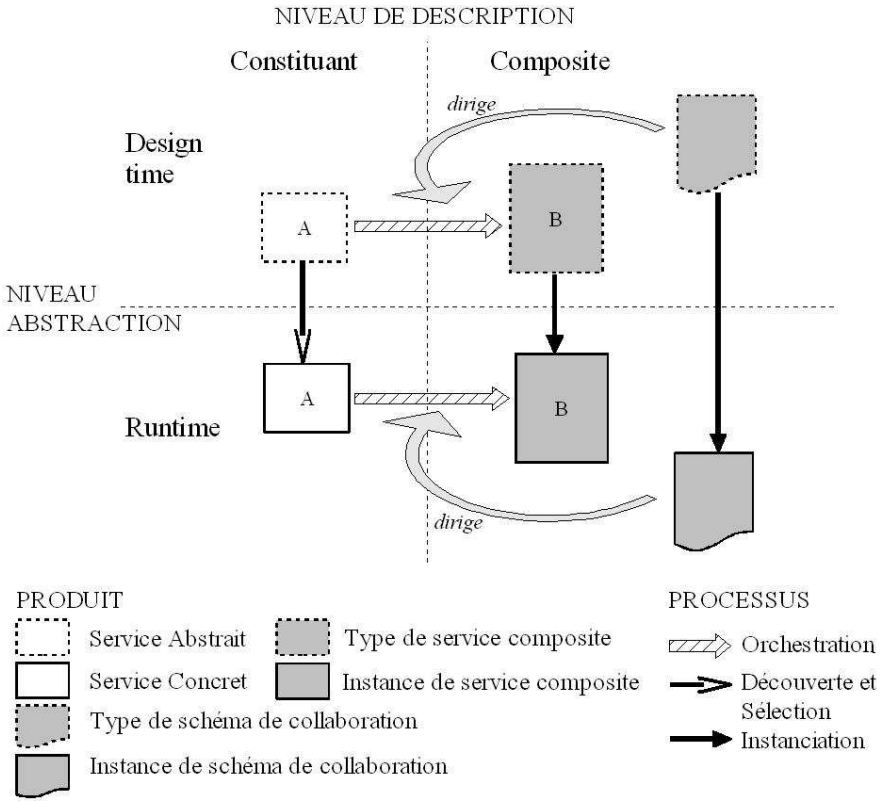

Fig. 4.6 – Répartition des produits et processus du méta-modèle de service composite

L'autre principale contribution du méta-modèle est la définition du processus d'*auto-composition*. L'auto-composition correspond à la gestion dynamique des interdépendances entre gestionnaires par le gestionnaire de service composite (GSC, section 2.2.2). Ce dernier coordonne leurs exécutions respectives afin de conserver la cohérence du service composite lors des adaptations contextuelles de sa composition de services. Il dirige ces adaptations en manipulant et modifiant l'ensemble des services métiers concrets réutilisés. Ainsi, l'auto-composition est un processus du runtime qui intervient entre les niveaux de description.

La table 4.3 fait le récapitulatif des produits et processus qui sont définis et utilisés par notre méta-modèle de service composite. Les éléments en ***italique*** soulignent les contributions du méta-modèle de service composite.

La figure 4.6 reprend la représentation de la figure 4.2 et illustre l'organisation entre produits et processus du méta-modèle de service composite. Le découpage en niveau de description se fait sur trois paliers, les services constituants, le schéma de collaboration, et le service composite qui encapsule ce schéma. Ce découpage rappelle celui des composants avec respectivement, les composants constituants, la configuration et le composant composite. Les processus de découverte et sélection de services assurent le passage du design-time au runtime des services abstraits vers les services concrets. Le processus d'instanciation assure le passage entre les types de schéma de collaboration et

Tab. 4.4 – Expression de la composabilité et de la dynamicité

Impacts $\alpha > \beta > \gamma$	α	β	γ
Composabilité	Abstraction de communication, Pouvoir évolutif.	Architecture explicite, Couplage faible.	Pouvoir expressif, Responsabilité propriétaire.
Dynamicité	Couplage faible, Pouvoir évolutif, Abstraction de communication.	Responsabilité propriétaire, Architecture explicite.	Pouvoir expressif.

de service composite vers les instances de schéma de collaboration et de service composite. Enfin le processus d'orchestration coordonne les invocations entre produits de niveaux de description différents et il n'y a pas de relations entre produits d'un même niveau de description.

4.6.2.2 Aspects qualitatifs

Les produits et processus précédents sont utilisés pour améliorer certains aspects qualitatifs du paradigme SOSE liés à la composition dynamique. Ce processus de composition est l'expression de deux des trois critères majeurs de qualités qui sont au coeur des préoccupations du SOSE : la *composabilité* et la *dynamicité*. Dans la section 4.5.3.2, nous avons défini *composabilité* et *dynamicité* suivant des combinaisons particulières des six propriétés. Le tableau 4.4 résume la répartition de ces propriétés suivant un ordre décroissant de leur impact sur les deux qualités, en groupes α, β et γ.

Tout d'abord, le principe du méta-modèle est d'expliciter la notion de service composite en organisant l'ensemble des aspects importants identifiés dans la littérature. Il définit de façon précise les entités qui interviennent dans le service composite et les relations entre ces entités. Ainsi, il renforce la propriété d'*architecture explicite* en proposant une architecturation claire du service composite. De plus, la gestion des hétérogénéités et la manipulation des types de schéma de collaboration par un possible mécanisme d'héritage renforcent l'*abstraction de communication* en fournissant aux architectes des nouveaux moyens pour mieux appréhender et réutiliser les communications entre services.

D'autre part, notre méta-modèle de service composite apporte une meilleure gestion des niveaux d'abstraction. Il explicite les entités architecturales qui composent ces niveaux : service abstrait, service concret, type de schéma de collaboration, etc. Puis, il clarifie les moyens de les manipuler : découverte et sélection de services, orchestration de services abstraits, etc. De plus, il propose une nouvelle notion de service composite et dresse le portrait du processus d'auto-composition. Ainsi, ces apports de nouveaux concepts qui sont mis à la disposition des différents acteurs du développement logiciel (architectes, designers, développeurs, etc.) améliorent le *pouvoir expressif* du SOSE.

Enfin, comme expliqué dans les chapitres précédents, le méta-modèle de service

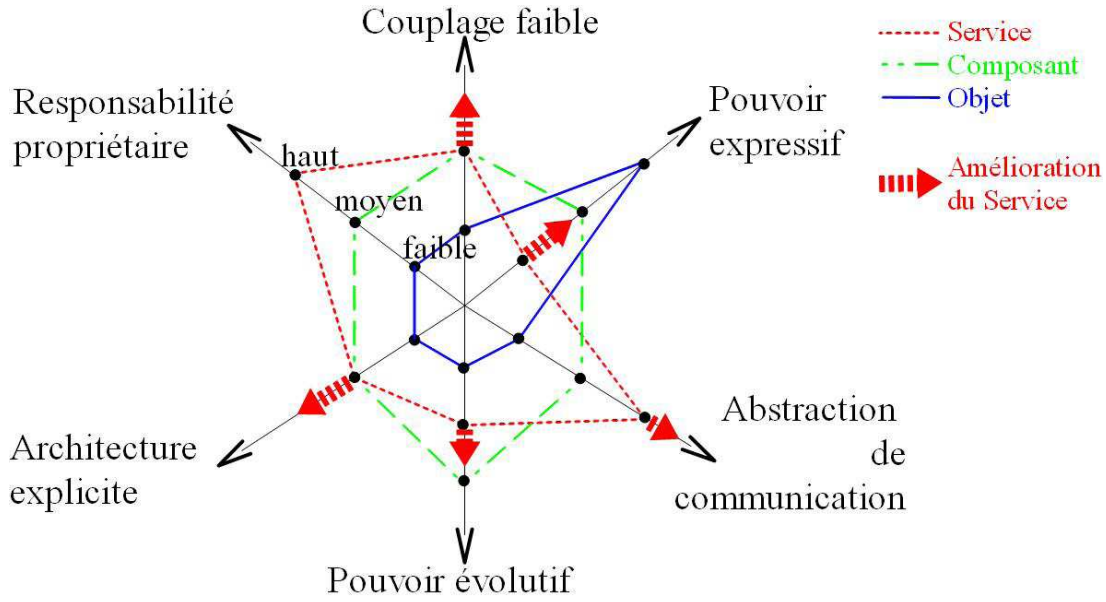

Fig. 4.7 – Synthèse des apports qualitatifs du méta-modèle de service composite

composite est construit avec une optique de réduction du couplage entre les services en collaboration. En effet, le processus d'auto-composition assure les évolutions contextuelles dynamiques de la composition de services. Le service composite est moins dépendant des services constituants qu'il réutilise. Le couplage entre les besoins du composite, exprimés par ses services abstraits, et les services concrets qui remplissent ces besoins est moins fort. Ainsi, l'auto-composition renforce deux propriétés qualitatives du SOSE : le *pouvoir évolutif*, de par son existence en tant que nouveau processus d'évolution d'une composition de services ; et le *couplage faible*, en réduisant la dépendance du composite envers ses services concrets constituants.

La figure 4.7 reprend l'évaluation qualitative du SOSE (figure 4.3) suivant nos six propriétés principales et illustre les améliorations ciblées par le méta-modèle de service composite.

Ainsi, notre méta-modèle de service composite, qui est une contribution à la problématique de composition dynamique de services, intervient sur les trois propriétés dont les impacts sont les plus importants sur la composabilité et la dynamicité (table 4.4) : l'*abstraction de communication*, le *pouvoir évolutif* et le *couplage faible*.

4.6.2.3 Vers une méta-modélisation par services

La définition du méta-modèle de service composite participe à la méta-modélisation par services afin de fournir au SOSE une solide base sémantique. La hiérarchie de méta-modélisation que nous utilisons est une pyramide composée d'exactement quatre niveaux d'architectures qui sont spécifiés dans les travaux précédents de notre équipe [LeG09, Ami10]. Cette pyramide est construite en parallèle avec les quatre niveaux de modélisation

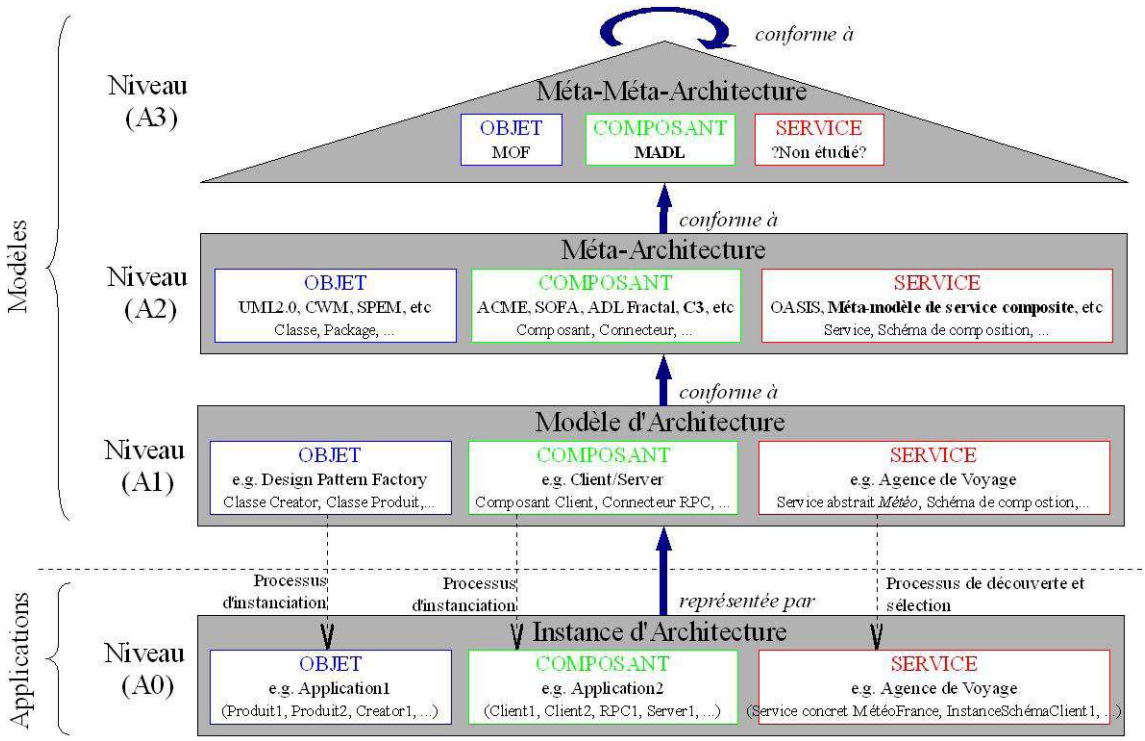

Fig. 4.8 – Hiérarchie de méta-modélisation composant et service

proposés par l'OMG. Au niveau le plus bas, la strate A0 correspond au système réel c'est-à-dire le niveau application/instance. Un modèle d'architecture représente ce système au niveau A1. Ce modèle est conforme à son méta-modèle du niveau A2. Ce méta-modèle est lui-même conforme à un méta-méta-modèle du niveau A3. Ce méta-méta-modèle est réflexif, c'est-à-dire qu'il est conforme à lui-même.

La figure 4.8 illustre ces quatre niveaux et leurs relations. Elle est directement reprises des travaux précédents de notre équipe [Ami10] et permet de localiser notre méta-modèle de service composite au niveau A2. De plus, elle identifie son positionnement par rapport aux autres contributions de l'équipe sur les méta et méta-méta architectures à composants que sont les modèles **C3** [Ami10] et **MADL** [SOK05].

Ainsi, cette figure fait le parallèle entre la méta-modélisation par services et la méta-modélisation par composants [Wil99, SOK05]. Elle met en évidence un point de différenciation cruciale entre composant et service sur le passage du niveau A1 au niveau A0 où l'instanciation d'une architecture à base de services inclut les processus de découverte et sélection dynamiques des services concrets réutilisés. De plus, elle souligne, à notre connaissance, le manque au niveau A3 d'un méta-méta-modèle dédié au service.

4.6.2.4 Amélioration du méta-modèle de service composite

Notre étude du service composite repose sur le principe de séparation entre type et instance de schéma de collaboration qui améliore leur réutilisabilité. Dans la section 2.2.3.3, nous avons déjà exposé le potentiel d'un processus d'*héritage entre types de schéma de collaboration* qui pourrait faciliter les spécialisations de comportements du service composite. Le fournisseur du service composite pourra mettre en place des fonctions personnalisées suivant les négociations avec un nouveau client sans toucher aux comportements déjà existants dédiés aux autres clients.

Dans cette continuité, nous pouvons chercher à formaliser ce processus d'héritage qui devra être étudié à trois niveaux distincts :

- sur le schéma de collaboration globale : de fait un type de schéma de collaboration est associé à un graphe, un héritage sur ce graphe statuerait des modifications autorisées sur la topographie de l'ensemble en termes d'ajouts de noeuds ou d'arcs ;
- sur les services abstraits : qui sont les noeuds du graphe ;
- sur les liens : qui sont les relations entre les services abstraits permettant de définir la collaboration globale.

En parallèle, la recherche autour de la composition de services et de la notion de service composite doit aussi étudier un possible processus de *composition de description de services* (section 1.4.4.2). L'objectif est de pouvoir automatiser le développement de la *description d'un service composite* à partir des descriptions des services concrets qu'il réutilise. En définissant ce processus, la publication du service composite sera plus efficace et plus rapide.

Un autre aspect du SOSE qui semble intéressant d'étudier est sa représentation au niveau A3 de la hiérarchie de méta-modélisation, comme souligné par la figure 4.8. À notre connaissance, il n'existe pas encore de travaux qui étudie un méta-méta-modèle dédié au service d'une façon analogue aux travaux autour de la définition d'un méta-méta-modèle composant [Wil99,SOK05]. La recherche autour d'un méta-méta-modèle service est encore à faire afin de déterminer son éventuel intérêt.

4.7 Conclusion

Dans ce chapitre, nous avons présenté notre cadre conceptuel de comparaison entre objet, composant et service. L'objectif premier de ce travail est de compléter le manque persistant laissé par la littérature autour de la spécification claire des différences conceptuelles entre un composant et un service. En outre, la prise en compte de l'OO a pour intérêt d'apporter une vision plus globale de l'évolution des préoccupations du génie logiciel.

Pour diriger notre analyse, nous avons choisi une approche top-down qui se focalise d'abord sur les aspects conceptuels des différents paradigmes avant de développer les qualités qui en découlent. Le choix des parties et sous-parties de ce cadre de comparaison cherche à le rendre le plus générique possible. Il se veut indépendant d'un paradigme de développement logiciel particulier afin d'éviter tout favoritisme. Nous cherchons ainsi

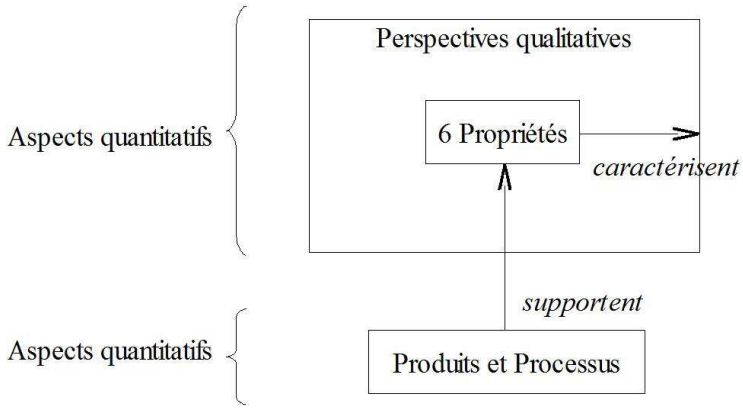

Fig. 4.9 – Fonctionnement global du cadre conceptuel de comparaison

à optimiser son potentiel de réutilisabilité et sa capacité d'application pour d'autres domaines du génie logiciel.

Ainsi, notre comparaison conceptuelle s'articule autour d'un tryptique *produit*, *processus* et *qualité*. Les produits et processus représentent la partie quantitative dans laquelle nous listons les concepts définis par les paradigmes OO, CBSE et SOSE qui sont nécessaires au développement d'applications nouvelles. Cette partie n'a pas vocation à l'exhaustivité de cette liste, mais cherche au contraire la minimalité des concepts tout en ciblant la complétude de la mise en évidence des spécificités de chacun. La partie qualitative définit six *propriétés* qui regroupent les éléments influençant la qualité du développement et des logiciels produits. L'évaluation des paradigmes suivant ces propriétés se base sur l'analyse quantitative précédente. Enfin, la partie qualitative permet aux utilisateurs d'exprimer leurs propres critères de qualité afin de comparer les approches mesurées par les six propriétés suivant leurs préoccupations individuelles. Cette capacité de personnalisation est fournie par la notion de *perspective qualitative* qui traduit l'expertise de l'utilisateur. De fait, une perspective lui permet de formuler sa compréhension d'une qualité cible à travers sa façon d'exploiter et de combiner les six propriétés.

La figure 4.9 résume le fonctionnement de notre cadre conceptuel de comparaison. L'aspect quantitatif répertorie les produits et processus des paradigmes respectifs qui sont les supports des six propriétés que nous avons identifiées. Ces propriétés caractérisent les critères de qualité et servent de vocabulaire aux utilisateurs pour exprimer leurs propres perspectives sur les qualités qui les préoccupent.

Au travers du cadre conceptuel de comparaison, nous avons ainsi présenté notre compréhension des différences entre composant et service. Nous avons proposé une évaluation qualitative en six propriétés (section 4.5.2) qui statue d'une certaine hiérarchie entre ces paradigmes.

En fin de chapitre, nous faisons un retour sur nos travaux de thèse autour du méta-modèle de service composite et de la notion de couplage faible. Nous mettons en évidence leurs apports en nouveaux produits et processus et leurs impacts sur l'évaluation de la

qualité de l'orienté service. Ce retour nous permet de faire la synthèse de nos contributions et pose les perceptives futures d'amélioration du paradigme SOSE.

De plus, nous listons le travail qui reste à accomplir sur le cadre en lui-même. Nous soulignons en particulier les aspects qualitatifs et le manque de formules d'évaluation objectives des cinq propriétés autres que le *couplage faible* (*responsabilité propriétaire*, *pouvoir évolutif*, *pouvoir expressif*, *architecture explicite* et *abstraction de communication*).

Ce chapitre met en évidence la double vocation du cadre conceptuel qui offre une méthode de comparaison entre différents paradigmes de développement mais aussi un moyen d'évaluer l'état actuel de chacun d'eux.

Chapitre 5

Réalisation et Expérimentation du méta-modèle

5.1 Introduction

L'intelligence ambiante (Iam) [Wei91] est une notion informatique qui met en oeuvre quatre principes de base :

- la capacité de l'utilisateur humain à interagir avec les différents appareils interconnectés de son environnement ;
- la transparence de cette interaction qui doit s'intégrer au quotidien des utilisateurs par des interfaces naturelles ;
- la faculté du système à connaître le contexte d'usage pour une application donnée en se basant sur la présence et le positionnement des ressources ou des personnes ;
- et donc, la capacité de ce système à s'adapter dynamiquement aux situations.

De tels environnements constituent un coeur de cible privilégié du SOSE de par la nature hautement volatile et hétérogène des ressources exploitées et des utilisateurs.

Dans ce chapitre, nous présentons une expérimentation de notre méta-modèle de service composite dans ce contexte Iam, à travers la simulation d'un environnement de domotique. La domotique correspond à l'appellation commerciale de l'application des principes Iam dans le cadre domestique, où la maison représente l'environnement, les différents appareils constituent les ressources hétérogènes (électroménagers, lumières, portes, capteurs, etc.), et les habitants en sont les utilisateurs principaux.

L'objectif est d'illustrer une application du méta-modèle afin d'appuyer la faisabilité de notre approche basée sur la répartition des logiques de composition dans les différents services gestionnaires et sur leur coordination dans le processus d'auto-composition. Ainsi, nous présentons un exemple de projection de ces concepts sur des technologies réelles, en l'occurrence SCA et Java.

Le chapitre est organisé de la façon suivante. La section 5.2 introduit le scénario de domotique envisagé. L'instanciation de notre méta-modèle par rapport à ce scénario est présentée dans la section 5.3. La section 5.4 fait ensuite la projection de cette instance sur les technologies SCA et Java. Elle décrit les propriétés de SCA et justifie le choix de cette plate-forme. Puis, elle présente les restrictions qui ont été prises entre le modèle et l'implémentation effective. Enfin, la section 5.5 conclut ce chapitre en revenant sur les limites de cette expérimentation et en soulignant les travaux d'implémentation futurs.

5.2 Scénarii de domotique

Un premier scénario de domotique appelé "écoute de musique" dérive de l'exemple des chapitres 1 et 2 auquel a été ajoutées des contraintes sur la gestion de multiples contextes utilisateurs. Un second scénario de "vidéo-conférence" est introduit pour mettre l'accent sur la gestion des concurrences dans l'exploitation des services disponibles.

5.2.1 Écoute de musique et vidéo-conférence

Le scénario "écoute de musique" correspond à un résident qui souhaite écouter de la musique sur haut-parleur et désire que cette musique l'accompagne à travers ses déplacements dans les différentes pièces de la maison. Tout d'abord, le résident exprime oralement son choix de musique via l'interface homme-machine la plus proche. Ensuite, le système récupére ce choix et recherche à travers les appareils stockant de la musique celui dont le répertoire répond à la demande. La musique est jouée et le son diffusé dans la salle où se situe l'utilisateur. Enfin, sa musique l'accompagne à travers ses déplacements, c'est-à-dire que le système éteint le haut-parleur dans la salle quittée et l'allume dans la salle où pénètre l'utilisateur.

Ce scénario met en scène une collaboration entre 3 appareils de la maison :
- l'interface capable de récupérer le choix de l'utilisateur ;
- l'appareil en charge de la lecture de musique ;
- l'appareil de diffusion du son.

Le scénario "vidéo-conférence" correspond à un résident qui reçoit un appel de vidéo-conférence qui correspond classiquement à une collaboration entre 3 ressources :
- l'appareil de réception-émission de la communication ;
- l'appareil de diffusion de l'image et du son ;
- l'appareil de récupération de l'image et du son.

Ce résident peut procéder à ce scénario dans toute salle possédant simultanément les appareils de diffusion et récupération de l'image et du son.

5.2.2 Contraintes d'exécutions

La gestion des deux scénarios implique un certain nombre de contraintes que nous avons posées pour constituer notre exemple. Elles sont réparties en deux catégories :

- contraintes simples : qui abordent le traitement d'un scénario classique ;
- contraintes complexes : qui traitent des interactions complexes entre plusieurs scénarios et plusieurs utilisateurs.

5.2.2.1 Contraintes simples

La première contrainte correspond à la **localisation de l'utilisateur**. De fait, ses déplacements à travers les pièces dirigent quelle interface homme-machine (typiquement un microphone pour la récupération du choix oral) et quel appareil d'émission de son doivent être utilisés. Le système doit être capable d'arrêter la diffusion de son dans les salles quittées par l'utilisateur tout en la démarrant dans celle où il pénètre. De la même façon, le changement d'interface doit être transparent pour l'utilisateur qui peut à tout moment modifier son choix musical. Ce nouveau choix peut alors modifier l'appareil de lecture utilisé en fonction de sa bibliothèque de titres musicaux.

Une autre contrainte est posée sur la **qualité des appareils** où le système devra chercher à offrir en priorité le meilleur service disponible, que ce soit dans la portée de réception du microphone pour éviter les déplacements de l'utilisateurs, ou sur l'acoustique des haut-parleurs et des lecteurs.

Enfin, le système doit gérer la **disponibilité des appareils** au moment de la demande de collaboration et être capable d'identifier des solutions alternatives pour assurer la continuité de service. La gestion de ces disponibilités est d'autant plus complexe dans un cadre multi-utilisateurs et multi-scénarios et représente les contraintes complexes.

5.2.2.2 Contraintes complexes

La gestion multitenant est un aspect essentiel des services SOSE. Cet aspect est modélisé dans notre exemple de domotique par la possibilité d'exécution de plusieurs scénarios en parallèle et par la demande de plusieurs clients pour un même scénario.

Dans ces deux cas, nous avons défini un jeu de priorités qui s'exprime sous différentes contraintes.

La première contrainte correspond à l'**interaction entre différents scénarios en exécution**. Cette interaction peut être la cause de changements d'état de disponibilité des ressources en cours d'utilisation ou utilisables du point de vue d'un scénario. Nous posons donc une hiérarchie dans l'importance des scénarios pour définir les priorités d'exécution suivant deux aspects :

- une priorité temporelle : le premier scénario arrivé est le premier servi ;
- une priorité relative : un scénario statué comme plus important que l'autre s'accaparera automatiquement la ressource (typiquement un scénario d'alerte incendie prendrait le pas sur notre scénario d'écoute de musique).

Une seconde contrainte est liée aux **rapports entre utilisateurs** où des priorités différentes sont accordées à chacun d'entre eux. Ces rapports prioritaires causeront des inégalités sur la disponibilité des appareils. L'exemple le plus représentatif dans un foyer domestique est le rapport entre parents et enfants où typiquement les premiers imposeraient leurs exécutions aux seconds. La gestion des priorités inclut aussi le principe de possession matérielle de la pièce ou de l'appareil. Par exemple, dans sa chambre l'enfant peut avoir une priorité supérieure aux parents de même que sur des ressources enregistrées comme lui appartenant (chaîne Hi-fi personnelle). Cette approche est généralisée à une notion de propriété d'un utilisateur par rapport à une salle ou une ressource.

Enfin, certains appareils sont capables de gérer plusieurs utilisateurs en parallèle. Dans notre simulation, un même microphone peut récupérer le son émis par plusieurs personnes. Par la suite, le système détermine quelle est la personne qui parle puis l'associer avec le scénario ciblé.

5.3 Instanciation du méta-modèle

Après avoir clairement posé les scénarios considérés, nous présentons leurs modélisations à travers notre méta-modèle de service composite. Nous nous attarderons plus spécifiquement sur l'exemple "écoute de musique" tout en faisant le parallèle avec la gestion de "vidéo-conférence", leurs services métiers utilisés et leurs collaborations étant relativement similaires. Dans notre modélisation, chaque scénario est ainsi associé à un service composite.

5.3.1 Service composite : Commande vocale

Dans notre application, nous supposons qu'un service composite *Commande vocale* est en permanence actif. Ce service fait l'interface entre l'utilisateur et son environnement et déclenche l'exécution des différents scénarios en fonction des demandes. Ces déclenchements de scénarios correspondent à l'invocation des services composites associés aux scénarios en leur fournissant les données d'entrées nécessaires. Ainsi, il fait appel au service composite *Écoute de musique* pour exécuter le scénario "écoute de musique", au service composite *Vidéo-conférence* pour exécuter le scénario "vidéo-conférence".

5.3.1.1 Fonctionnement métier

Dans notre modélisation, le service composite *Commande vocale* utilise quatre services métiers :

- *Microphone* : service métier qui représente l'ensemble des microphones actifs en permanence et positionnés dans la maison.
- *Analyse_vocale* : service métier qui extrait des sons récupérés la demande de l'utilisateur en terme de scénario.
- *Écoute de musique* : service métier qui exécute le scénario "écoute de musique".
- *Vidéo-conférence* : service métier qui exécute le scénario "vidéo-conférence".

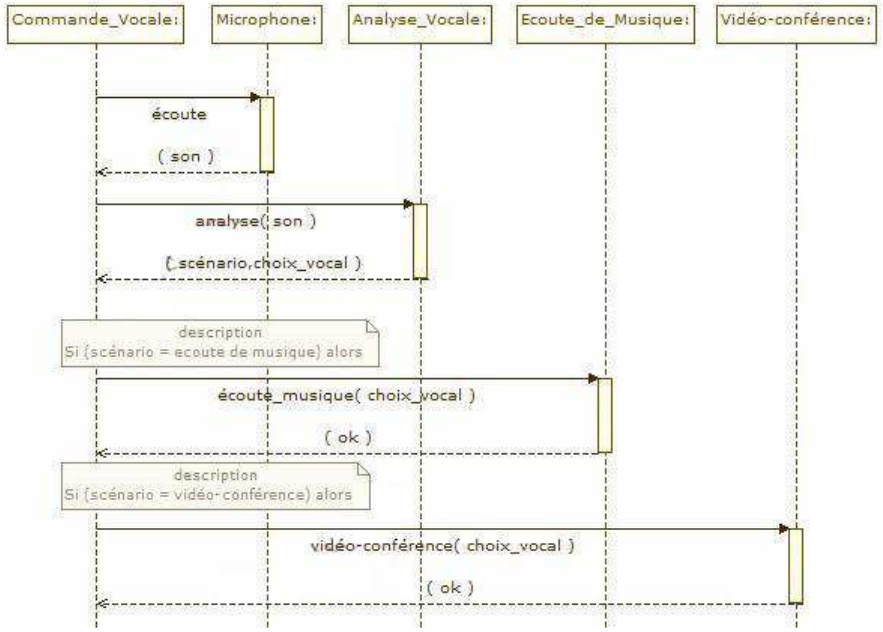

Fig. 5.1 – Diagramme de séquence : fonctionnement du service composite *Commande vocale*

La collaboration entre ces services métiers, représentée sous forme du diagramme de séquence de la figure 5.1, se définit de la façon suivante : le service *Microphone* récupère les sons émis par les utilisateurs et les transmet au service *Analyse_vocale*. Ce dernier étudie le flux audio et en extrait la commande d'ordre qui identifie le scénario désiré. Enfin, le scénario adéquat est déclenché par invocation des services *Écoute de musique* ou *Vidéo-conférence* en leur fournissant les bonnes données. Pour le scénario "écoute de musique", la donnée d'entrée correspond au choix oral d'une musique ou une chanson. Pour le scénario "vidéo-conférence", la donnée d'entrée est l'identification orale de la personne avec qui l'utilisateur souhaite avoir la conversation.

5.3.1.2 Modélisation du service composite Commande vocale

La figure 5.2 décrit une modélisation simplifiée du service composite *Commande vocale*. Il possède un GSC et un SCM. Le SCM regroupe les quatre services métiers concrets : *Microphone*, *Analyse_vocale*, *Écoute_de_ musique* et *Vidéo-conférence*. Le GSC regroupe les différents services gestionnaires :

– le gestionnaire de collaboration : qui est composé du gestionnaire d'observation qui possède un générateur de moniteurs, un moniteur de classe pour l'observation de la fonctionnalité de commande vocale (*H1*) et autant de moniteurs d'instances qu'il y aura d'exécutions de cette fonctionnalité (dans notre représentation, il y a une instance *Clt1* et un moniteur d'instance associé *MI1*). De plus, le gestionnaire de collaboration possède un moteur d'exécution pour exécuter les différentes instances

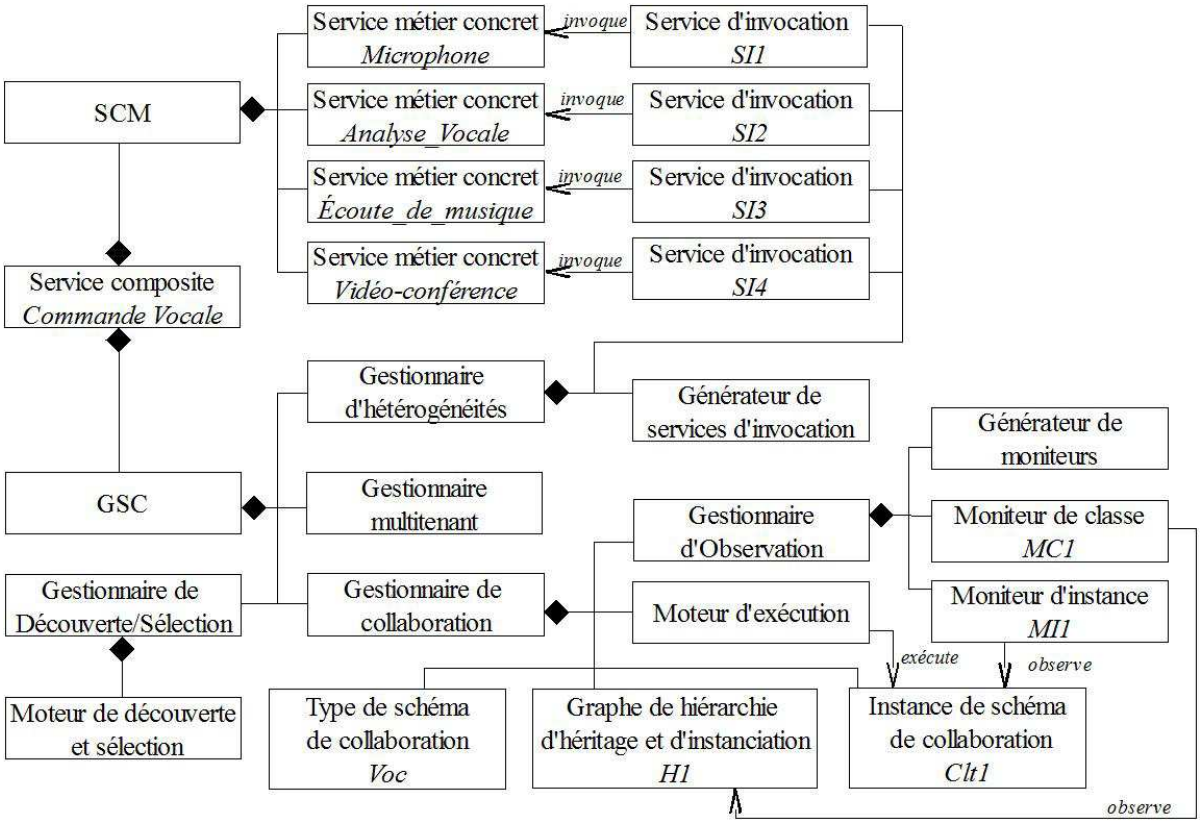

Fig. 5.2 – Modélisation du service composite *Commande vocale*

de schéma de collaboration (telle que *Clt1*) ;

- le gestionnaire d'hétérogénéités : qui est composé d'un générateur de services d'invocation et des quatre services d'invocation générés (*SI1*, *SI2*, *SI3* et *SI4*) pour communiquer avec les quatre services métiers ;
- le gestionnaire de découverte et sélection : qui est associé à un moteur de découvertes et sélections de services pour le remplacement des quatre services métiers (*Microphone*, *Analyse_vocale*, *Écoute de musique* et *Vidéo-conférence*) en cas de défaillances.
- le gestionnaire multitenant : qui gère les différents contextes clients. Par exemple, on peut imaginer des commandes vocales dont l'exécution est restreinte à certains utilisateurs tel qu'un scénario de cinéma pouvant être déclenché uniquement par les parents afin de limiter son utilisation par les enfants.

La figure 5.3 décrit le service composite *Commande vocale* à travers les différents graphes maintenus par les services gestionnaires en reprenant le design des figures de la section 2.4. Ainsi, le fonctionnement métier du service composite *Commande vocale* est modélisé par un graphe de hiérarchie d'héritage et d'instanciation nommé *H1* (Figure 2.3). *H1* est composé de deux noeuds : *Voc* pour le type de schéma de collaboration qui décrit les coordinations entre les quatre services métiers abstraits (*SA1* à *SA4*), et *Clt1* une instance

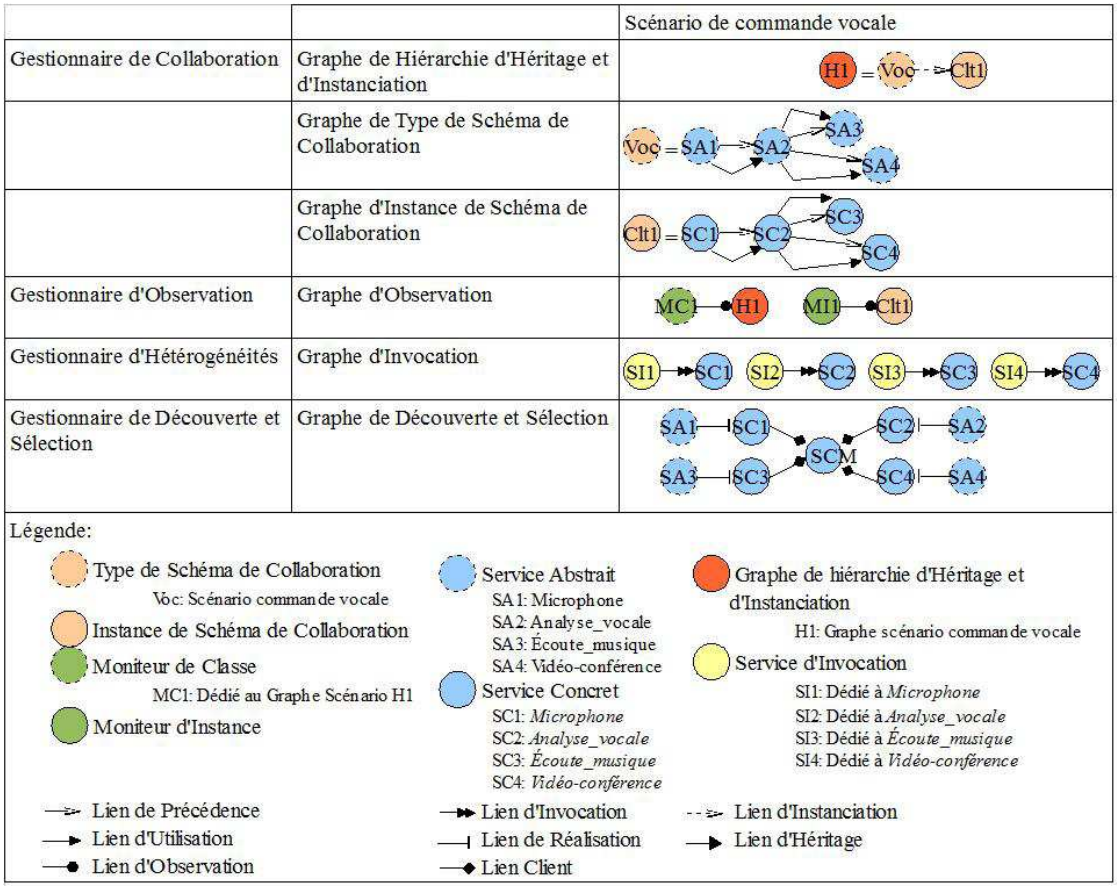

Fig. 5.3 – Service composite *Commande vocale* : graphes des services gestionnaires

de *Voc* créée à partir des services concrets (*SC1* à *SC4*) découverts et sélectionnés pour réaliser les quatre services abstraits précédents. C'est cette instance *Clt1* qui est exécutée par le gestionnaire de collaboration pour supporter le système de commande vocale. Dans le gestionnaire d'observation, un moniteur de classe *MC1* est associé à *H1*, et un moniteur d'instance *MI1* est associé à *Clt1*. Dans le gestionnaire d'hétérogénéités, chaque service métiers concrets (de *SC1* à *SC4*) est associé à un service d'invocation (de *SI1* à *SI4*). Enfin, le gestionnaire de découverte et sélection maintient les relations de réalisation où le service abstrait *SA1* est réalisé par le service concret *SC1*, et ainsi de suite jusqu'à *SA4*.

Enfin, la figure 5.4 décrit à nouveau le fonctionnement métier précédent (figure 5.1) en prenant en compte les différents services gestionnaires. Le principe est que le GSC fait appel au gestionnaire de collaboration pour l'exécution de l'instance de schéma de collaboration qui coordonne les services métiers concrets *Microphone*, *Analyse_vocale*, *Écoute de musique* et *Vidéo-conférence*. Lorsque cette exécution requiert des invocations aux services métiers réutilisés, le GSC intercepte ces messages vers l'extérieur puis les transmet au gestionnaire d'hétérogénéités. Le gestionnaire d'hétérogénéités invoque alors les services d'invocation adéquats qui sont en charge de la communication réelle avec les

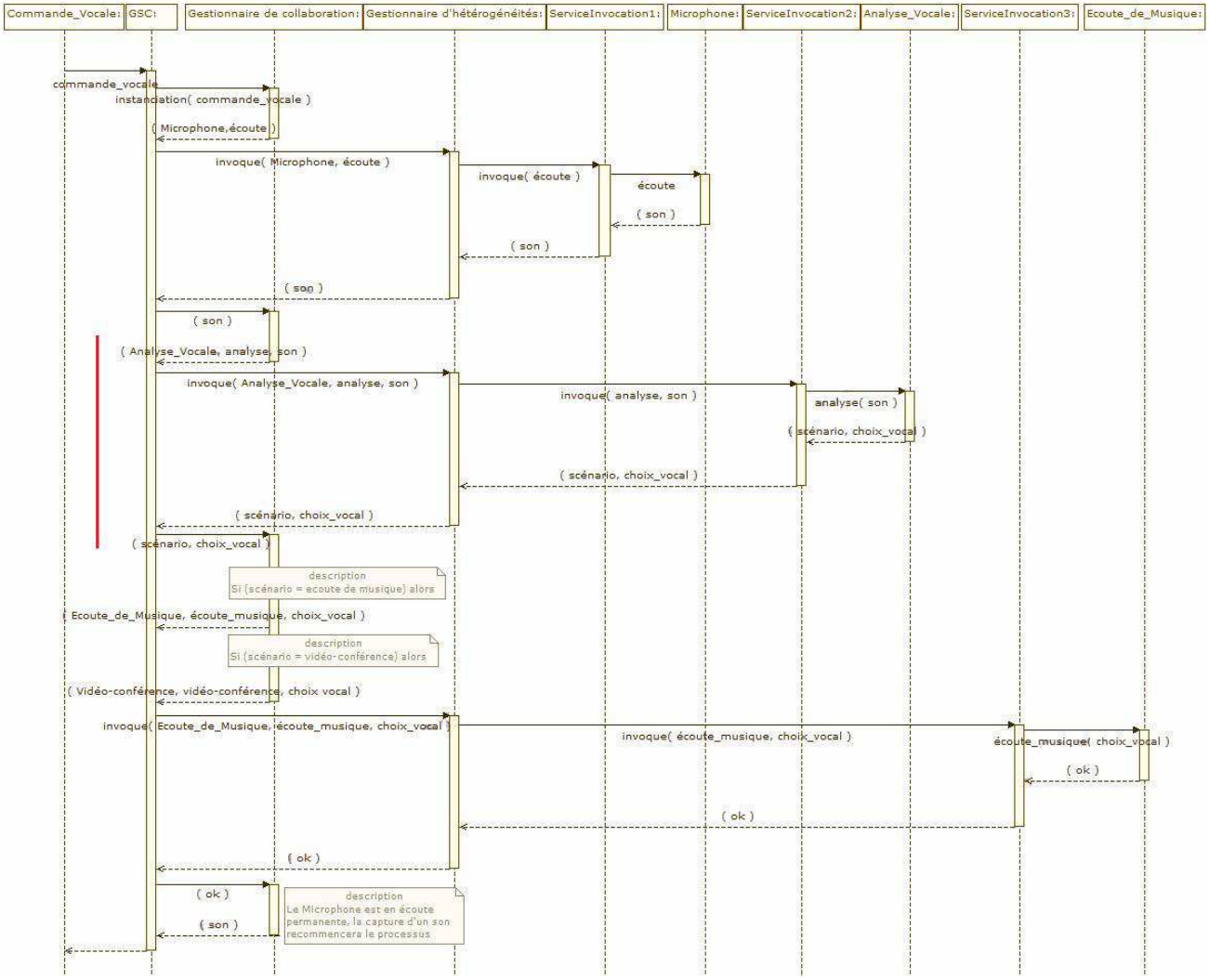

Fig. 5.4 – Diagramme de séquence : fonctionnement du service composite *Commande vocale* avec services gestionnaires

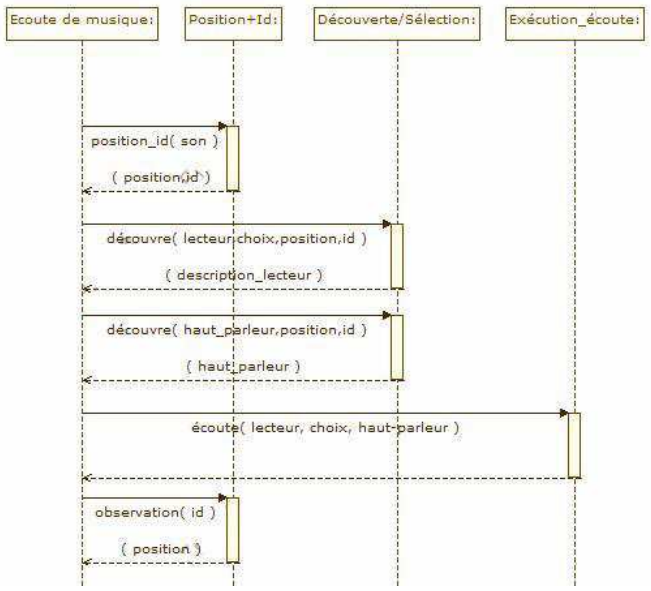

Fig. 5.5 – Diagramme de séquence : fonctionnement du service composite *Écoute de musique*

services métiers concrets.

Dans la figure 5.4 (barre verticale rouge), le gestionnaire de collaboration qui renvoie le triplet *(Analyse_Vocale, analyse, son)* indique quel service métier invoqué (*Analyse_Vocale*), quelle fonctionnalité de ce service est ciblée (*analyse*), et avec quelles données d'entrée (*son*). Le GSC demande au gestionnaire d'hétérogénéités d'invoquer ce service métier (*invoque(Analyse_Vocale, analyse, son)*) qui invoque à son tour le service d'invocation *ServiceInvocation2* en communication avec le service métier *Analyse_Vocale*. Le résultat de cette invocation fait le chemin inverse jusqu'au gestionnaire de collaboration qui peut continuer l'exécution.

5.3.2 Service composite : Écoute de musique

5.3.2.1 Fonctionnement métier

Le service composite *Écoute de musique* permet à un utilisateur, qui en fait la demande via le service composite *Commande vocale*, de lire une musique choisie sur un lecteur de la maison et de l'écouter sur le haut-parleur le plus près de sa position. Il utilise trois services métiers :

- *Position+Id* : service métier en charge la récupération de la position et de l'identifiant du client qui est à l'origine de la commande ;
- *Découverte/Sélection* : service métier qui se charge de découvrir les lecteurs et haut-parleurs satisfaisant à la commande et la position du client ;
- *Exécution_écoute* : service métier qui se charge de la diffusion de la musique.

La collaboration entre ces services métiers, représentée sous forme du diagramme de séquence de la figure 5.5, se définit de la façon suivante : le service métier *Position+Id* récupère la demande de l'utilisateur. Il effectue une analyse vocale pour déterminer l'identité de la personne. Grâce à cette information, il peut effectuer une recherche sur la position courante de cette personne. Enfin, il transmet au service métier *Découverte/Sélection* les données sur l'identité de la personne, sa position courante et le choix de musique.

Découverte/Sélection utilise ces informations pour déterminer l'appareil de lecture de musique possédant le chanson voulue et le haut parleur correspondant à la position. Il prend en compte les disponibilités des appareils en fonction des priorités liées à l'identité de l'utilisateur. Enfin, il transmet au service métier *Exécution_écoute*, le résultat de ces découvertes.

Dès lors, ce service *Exécution_écoute* fait appel au bon appareil de lecture qui lance la musique choisie, puis il transmet le flux audio au haut-parleur cible.

Enfin, le service composite *Écoute de musique* lance une routine demandant au service métier *Position+Id* de lui notifier tous changements de position de l'utilisateur. En cas de changement de salle, il fera à nouveau appel aux services *Découverte/Sélection* et *Exécution_écoute* pour identifier et exécuter les appareils adaptés à cette nouvelle position.

5.3.2.2 Modélisation du service composite Écoute de musique

La figure 5.6 décrit le service composite à travers les différents graphes maintenus par les services gestionnaires. D'une façon similaire à ce qui est présenté pour le service composite *Commande vocale*, le fonctionnement métier du service composite *Exécution_écoute* est représenté par un graphe de type de schéma de collaboration (Mus) maintenu dans le gestionnaire de collaboration. Lors d'une invocation du service composite, ce schéma est instancié (Clt1) avec les services métiers concrets SC1, SC2, SC3 dévolus aux rôles de *Position+Id* (SA1), *Découverte/Sélection* (SA2) et *Exécution_écoute* (SA3) pour supporter l'exécution réelle. Chacun de ces services concrets est associé à son service d'invocation (respectivement SI1, SI2, SI3) pour garantir les communications. Enfin, des moniteurs d'instance (MI1) et de classe (MC1) sont respectivement associés à l'instance (Clt1) de schéma de collaboration et au graphe de hiérarchie d'héritage et d'instanciation (H1 : composé de Mus et Clt1). Ces derniers observent les exécutions pour notifier tous besoins en adaptations en cas de services métiers défaillants. Dans ce cas précis, le processus d'auto-composition sera déclenché pour identifier, via le gestionnaire de découverte et sélection, de nouveaux services métiers concrets, puis garantir leur intégration correcte avec l'ensemble du composite.

Bien que le gestionnaire de découverte et sélection réutilise un moteur de découverte et sélection de services, il ne faut pas le confondre avec le service métier *Découverte/Sélection*. Ce dernier est une partie intégrante de la fonctionnalité métier demandée au service composite *Écoute de musique*, tandis que le gestionnaire de découverte et sélection est en charge de l'identification de nouveaux services métiers concrets lors de défaillances.

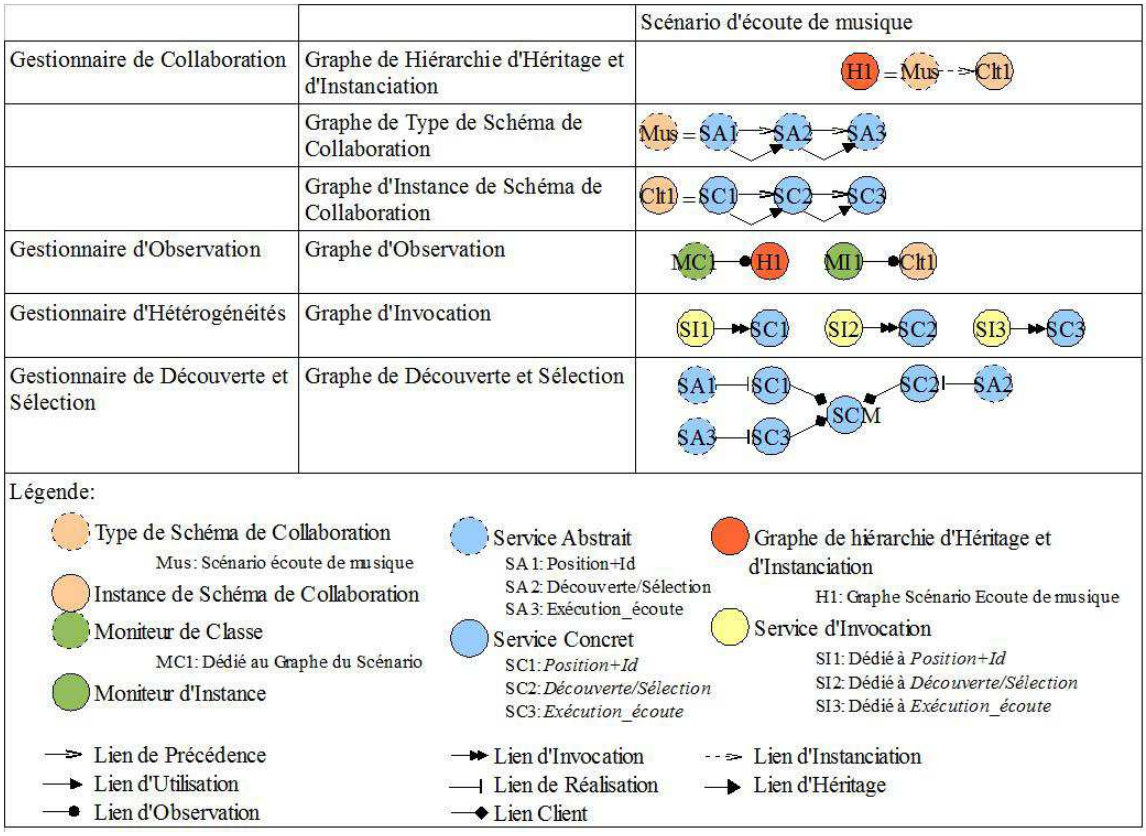

Fig. 5.6 – Service composite *Écoute de musique* : graphes des services gestionnaires

Ainsi, le gestionnaire de découverte et sélection peut être amené à trouver une solution alternative pour la réalisation du service métier *Découverte/Sélection*.

5.3.2.3 Service composite : Exécution_écoute

Le service métier *Exécution_écoute*, utilisé par le service composite précédent *Écoute de musique*, est construit comme un service composite qui est en charge des collaborations entre des services de lecture de musique et de haut parleur. Pour remplir son rôle, il utilise donc deux services métiers :

- *Lecteur* : service métier en charge de la lecture d'un fichier audio ;
- *Haut-parleur* : service métier en charge de la diffusion de son.

La collaboration entre ces services métiers abstraits est très simple, le *Lecteur* transmet le flux audio au *Haut-parleur* qui diffuse le son. Ainsi, le service composite *Exécution* prend en entrées les choix en services métiers concrets pour ses services abstraits *Lecteur* et *Haut-parleur*. Puis, il demande à son gestionnaire de collaboration d'instancier le type de schéma de collaboration en fonction de ces choix. Enfin, l'instance de schéma est exécutée.

5.3.3 Service composite : Vidéo-conférence

Nous construisons le service composite *Vidéo-conférence* sur le même principe que le service composite *Écoute de musique*. Ainsi, il réutilise trois services métiers :

- *Position+Id* : service métier en charge de la récupération de la position et de l'identifiant du client qui est à l'origine de la commande ;
- *Découverte/Sélection* : service métier qui se charge de découvrir les appareils responsables des communications vers l'extérieur, de la récupération d'image et de son, et de la diffusion d'image et de son ;
- *Exécution_vidéo* : service métier qui se charge de l'exécution de la vidéo conférence.

La collaboration entre ces trois services métiers est identique à celle présentée précédemment pour le scénario écoute de musique. Les deux premiers services métiers sont les mêmes que ceux utilisés par le service composite *Écoute de musique*. Enfin, le service métier *Exécution_vidéo* est un service composite avec un fonctionnement similaire au service métier *Exécution_écoute*. Il prend en entrées les services concrets identifiés par *Découverte/Sélection* et exécute la vidéo-conférence.

5.4 Réalisation en SCA et Java

Les technologies utilisées pour la réalisation de cette simulation sont SCA et Java. Dans cette section, nous faisons une brève présentation de SCA et nous discutons du choix de cette technologie. Puis, nous abordons les restrictions posées entre le modèle de la section précédente et l'implémentation réelle. Enfin, nous présentons quelques éléments de cette implémentation.

5.4.1 Présentation de SCA

SCA (Service Component Architecture) [OAS09] est un ensemble de spécifications qui décrit un modèle pour construire des applications et des systèmes pouvant être facilement intégrés dans une architecture orientée services. Il repose sur un modèle CBSE pour la construction de nouveaux systèmes comme des assemblages de composants. La figure 5.7 présente un exemple de système SCA construit avec l'outil Eclipse pour SCA. Le système est construit comme une composition de composants SCA. Un composant est constitué de services, de références et de propriétés non fonctionnelles. Les services exposent les fonctionnalités offertes, et les références exposent les fonctionnalités requises. Une composition de composants SCA se fait par l'établissement de liaisons (wire ou binding) entre les services et les références compatibles. Le modèle SCA supporte la notion de composant composite et de construction hiérarchique (un composant est implémenté par un composite). Enfin, le modèle SCA supporte la promotion de services, de références ou de propriétés se trouvant dans la composition qui peuvent être exposées au niveau du composant SCA encapsulant.

La figure 5.7 montre un exemple d'assemblage SCA. Un composite SCA (*Composite1*) est composé de deux composants SCA (*Composant1* et *Composant2*) reliés entre eux au

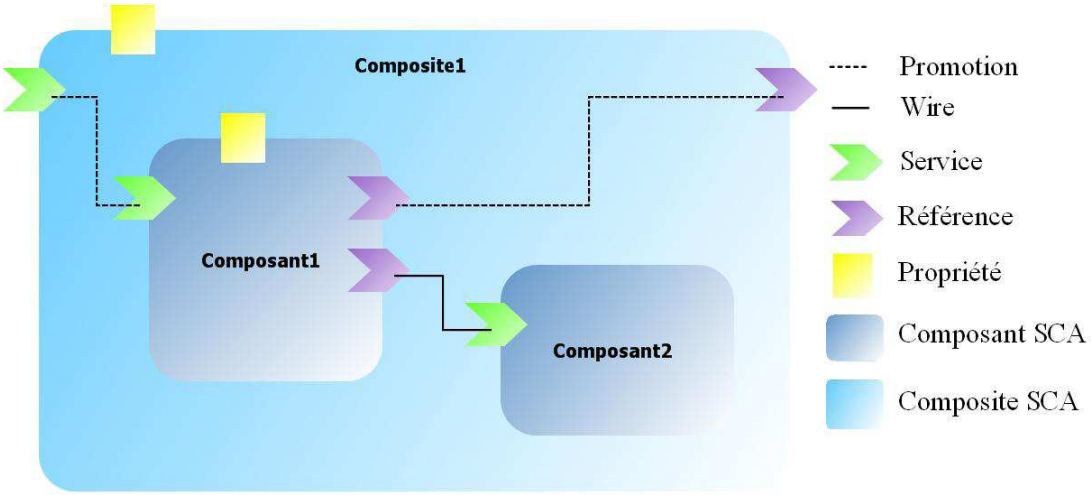

Fig. 5.7 – Service composite SCA

niveau d'un service. Enfin, un service et une référence de *Composant1* sont promus dans l'interface extérieure du composite *Composite1*.

Le principal avantage de SCA est que son modèle architectural ne fait pas d'hypothèse sur les technologies d'implémentations de ces différents éléments. Il offre une variabilité sur cinq niveaux : l'implémentation du composant SCA, l'implémentation de l'interface du composant (services et références), les protocoles de communication pour supporter les liaisons entre services et références, les propriétés non fonctionnelles, et le paquetage de l'application. Chacun de ces niveaux peut être réalisé par des technologies différentes et SCA garantira leur compatibilité et leur collaboration pour réaliser l'application globale.

Dans l'implémentation Tuscany[1] actuelle de SCA, utilisée pour la réalisation de notre simulation, l'outil permet de supporter des collaborations transparentes entre les cinq niveaux de variabilité pour les technologies suivantes :

- implémentation des composants : Java, C++, BPEL, Spring, langages de script, SCA composite ;
- implémentation des interfaces : WSDL, Java Interface ;
- protocoles de communication : SOAP, IIOP ;
- propriétés non fonctionnelles : security, transaction, etc.
- paquetage : ZIP, WAR.

Cette liste est non exhaustive et c'est cette gestion transparente des technologies qui a motivé le choix de SCA. De fait, elle s'identifie directement à notre gestionnaire d'hétérogénéités et simplifie son implémentation. De plus, SCA supporte les technologies services web WSDL et BPEL pour garantir l'orchestration et le principe fondamental du paradigme service qu'est l'exploitation de ressources déjà déployées.

[1] http ://tuscany.apache.org/

5.4.2 Restrictions par rapport aux modèles

Dans son implémentation actuelle, notre simulation de scénarios de domotique pose les restrictions suivantes par rapport à la modélisation précédente :

- *simulation du gestionnaire d'hétérogénéités* : l'utilisation de SCA permet la gestion des hétérogénéités technologiques (section 2.2.4) et assure le rôle des services d'invocation. Les hétérogénéités de types conservationnels et de données ne sont pas traitées.
- *absence du gestionnaire d'observation* : les défaillances sur les services métiers sont simulées manuellement par des modifications d'un paramètre lié à chaque ressource qui détermine son état. Ces déclenchements volontaires entraînent directement le processus d'auto-adaptation, il n'y a donc pas d'entités chargées des observations de contexte.
- *simulation du gestionnaire de collaboration* : le processus d'orchestration est simulé par des classes java et non par une technologie dédiée telle que BPEL. De fait, BPEL est une technologie complexe qui trouve son intérêt dans des exécutions asynchrones couplées à de la compensation. Dans notre exemple très simple, l'utilisation d'appels de méthodes java classiques est suffisante.
- *simulation du gestionnaire de découverte et sélection* : pour supporter les processus de découvertes et sélections, chaque ressource du système est associée à une description formalisée. Nous avons défini un vocabulaire simple pour classer les différents appareils présents (lecteur, haut-parleur, etc.). L'algorithme utilisé est de type 1-1 avec des correspondances fortes (section 3.5.1.2) entre les services abstraits et les services concrets et exploite un registre qui regroupe l'ensemble des services disponibles.
- *absence du gestionnaire multitenant* : les différents tests effectués sont restreints à un utilisateur unique.

La figure 5.8 est une capture d'écran du service composite *Écoute de musique* qui est associé aux trois services métiers *Position+Id* (*PositionServiceCmp*), *Découverte/Sélection* (*DiscoveryAndSelectionCMP*) et *Éxécution_écoute* (*ExecutionServiceCmp*) et illustre les restrictions précédentes. Ainsi, le rôle du gestionnaire de collaboration est réparti à travers les différentes liaisons entre les services et les références des composants SCA. Le principe d'orchestration est simulé par le composant *MusicPlayerSCMComponent* qui centralise les appels. Bien que nous ayons statué précédemment que le gestionnaire de découverte et sélection et le service métier *Découverte/Sélection* sont deux entités distinctes dans la modélisation, dans l'implémentation ils sont confondus derrière le composant SCA *DiscoveryAndSelectionCmp*.

5.4.3 Implémentation

Dans l'état actuel de l'implémentation, deux séquences ont été testées dans le cadre du scénario écoute de musique :

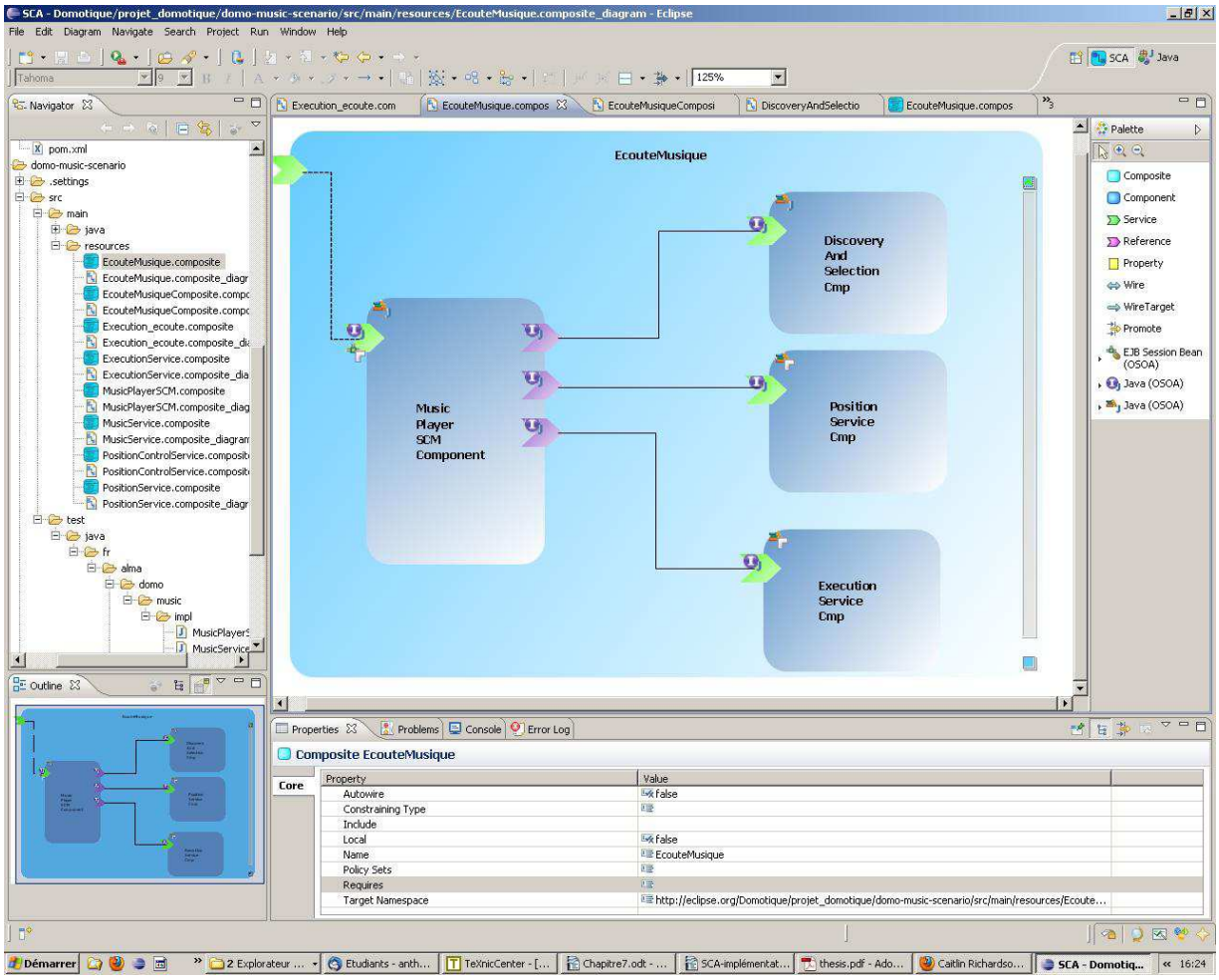

Fig. 5.8 – Capture d'écran : implémentation SCA du service composite *Écoute de musique*

- *séquence de déplacement* : l'environnement de domotique est initialisé avec deux salles, un haut-parleur dans chaque salle, un lecteur et un utilisateur. L'utilisateur est positionné dans la cuisine et lance le scénario d'écoute de musique. Puis, nous déplaçons manuellement l'utilisateur dans les WC et on vérifie que la diffusion du son s'est bien arrêté dans la cuisine et a repris dans les WC. (Figure 5.9)
- *séquence de défaillance* : l'environnement de domotique est initialisé avec une salle, deux hauts-parleurs, un lecteur et un utilisateur. L'utilisateur déclenche le scénario d'écoute de musique et un premier haut-parleur est utilisé. Ensuite, nous déclarons manuellement cet haut-parleur comme défaillant. La routine d'adaptation est déclenchée et remplace cet haut-parleur par le second disponible. (Figure 5.10)

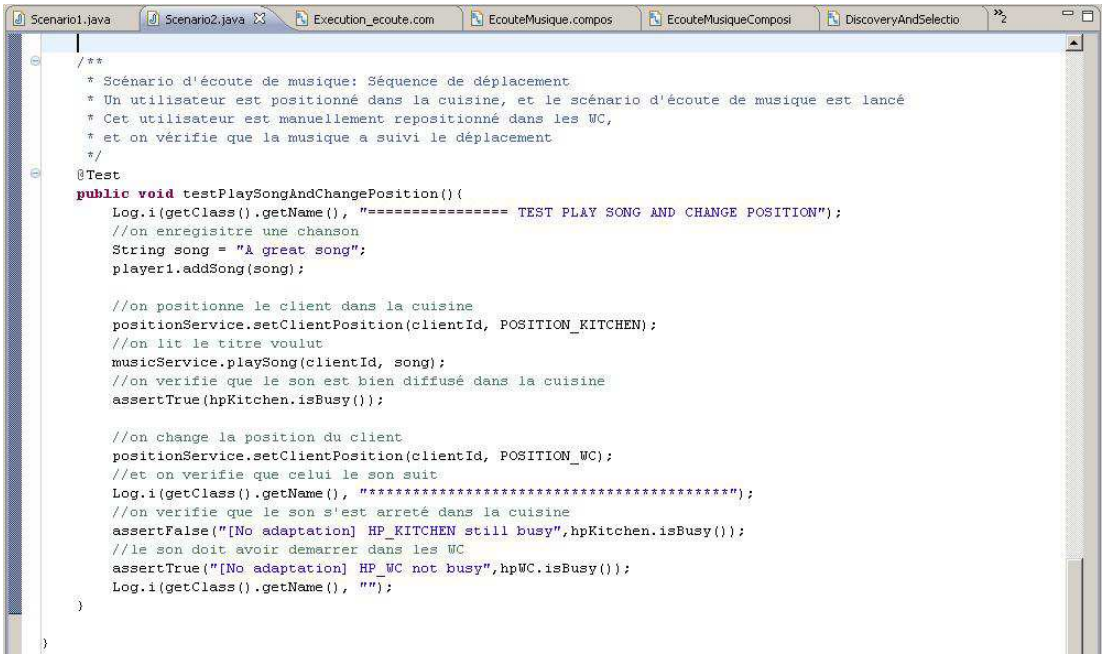

Fig. 5.9 – Capture d'écran : séquence de déplacement

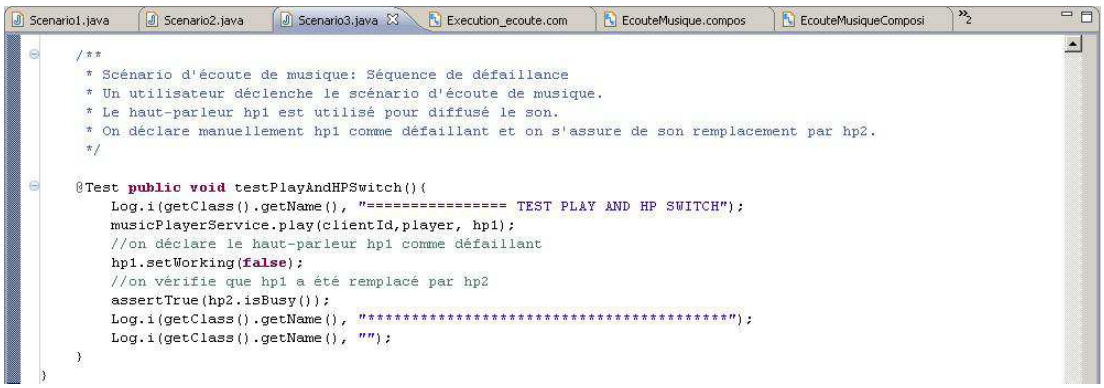

Fig. 5.10 – Capture d'écran : séquence de défaillance

5.5 Conclusion

Dans ce chapitre, nous avons présenté un environnement de domotique qui est modélisé à travers différents services composites respectant notre méta-modèle. Nous avons ensuite détaillé leurs imbrications et leurs fonctionnalités métiers. L'objectif est d'offrir un complément d'explication sur la vision proposée par notre service composite et d'illustrer quelques possibilités de projections des services gestionnaires vers des technologies particulières.

Dans l'état actuel de l'implémentation, le prototype reste du domaine du cas d'école et ne permet qu'une validation partielle du fonctionnement global du processus d'auto-composition. Ce travail a eu l'avantage de nous familiariser avec une technologie prometteuse qu'est SCA puis laissera la place, dans un futur proche, à un prototype plus ambitieux. De fait, nous voulons passer de notre statut d'utilisateur de SCA à celui de contributeur à la technologie en modifiant une de ces implémentations actuelles open source. Pour cela, nous pensons passer de la version classique Tuscany à la version FraSCAti [SMF+09] soutenue par l'INRIA. Cette dernière apporte des améliorations significatives (annexe A) en terme de technologies supportées mais surtout en terme de dynamicité et de reconfigurabilité. En effet, FraSCAti introduit la notion de reflexivité dans le modèle SCA qui permet des modifications à l'exécution telles que des ajouts ou des retraits de composants SCA ou de liaisons (wires) entre services et références. Nous souhaitons profiter de ces fonctionnalités pour organiser ces modifications dans le cadre de notre processus d'auto-composition. La finalité serait d'offrir aux utilisateurs de SCA un nouveau composant composite pour construire leurs applications. Cette nouvelle brique supporterait notre vision de la composition dynamique.

Conclusion et perspectives

Le paradigme service est devenu en une dizaine d'années seulement un courant majeur de l'ingénierie logicielle, à la hauteur des paradigmes objet ou composant. Il est soutenu par une communauté importante et très dynamique qui s'exprime, sur le plan national et international, à travers de nombreuses conférences, journaux ou ateliers dédiés. Ses succès académiques et industriels viennent de la nature de son approche et de son coeur de cible. Sa définition part d'un besoin de méthodes pour le développement rapide d'applications devant supporter des environnements hautement volatiles et hétérogènes. De tels environnements sont maintenant omniprésents, à travers une prédilection des nouveaux logiciels pour la réutilisation et la collaboration via les réseaux, ou encore, face aux intentions persuasives et ubiquitaires de l'intelligence ambiante. Le niveau de dynamicité et de transparence exigé pose de nombreuses problématiques dont le SOSE cherche à apporter des solutions. Ainsi, il a proposé un nouvel artefact de construction, le service, qui est à la base de nouveaux processus de développement logiciel.

La mise en oeuvre de ces processus est extrêmement complexe, où chacun d'eux se ramifie en des multitudes de problématiques sous-jacentes. Cette complexité a entraîné une explosion désorganisée de propositions se focalisant sur leurs nombreuses facettes. Face à ce constat qui se répète dans le SOSE en général, Erickson et Siau [ES08] ont exprimé le besoin de consensus sur les définitions apportées pour définir un unique modèle service.

Cette thèse s'inscrit dans ce mouvement où nous nous sommes attachés à mieux organiser certains aspects du paradigme service pour en apporter une meilleure compréhension et développer notre propre solution.

Cette conclusion revient sur ces différents aspects étudiés, puis présente les limitations de nos propositions qui s'ouvrent en autant de perspectives.

Bilan

Dans le chapitre 1, nous avons évoqué les principaux concepts et mécanismes du SOSE et tenté d'en éclaircir le fonctionnement. Nous nous sommes particulièrement intéressés à la notion d'architecture orientée services et au processus de composition de services. De fait, le SOSE repose sur un style particulier d'architecture qui supporte la décomposition d'une application en un ensemble de services en collaboration. Chaque service est une

ressource associée à un fournisseur qui le met à disposition des consommateurs potentiels via des registres. Cette organisation implique que consommateurs et fournisseurs ne se connaissent pas à priori. L'établissement dynamique de l'utilisation d'un service repose sur deux processus : la publication de services et la composition de services. Ce processus de composition, sujet privilégié de notre état de l'art, est en charge de l'identification des services utiles au développement de l'application, puis de leur collaboration effective. Plus tard, c'est à travers cette perspective de composition que nous avons mis en avant l'importance d'une coopération de la recherche pour atteindre un consensus salvateur à la résolution d'une problématique si complexe.

Dans la continuité de la recherche de ce consensus, nous proposons, dans le chapitre 2, un méta-modèle de service composite qui exprime et organise dans un seul modèle un certain nombre d'aspects importants de la composition dynamique de services. Notre postulat de départ est que la très grande diversité des approches existantes n'est pas due à une mauvaise analyse du processus de composition, mais est le reflet de sa complexité, chacune d'elles apportant sa contribution sur un aspect particulier et participe à la résolution du problème global. Notre méta-modèle de service composite se focalise sur la réduction du couplage entre les services constituants et organise les éléments qui influencent cette propriété. La spécification de leur collaboration permet ensuite de définir le processus d'auto-composition qui repousse plus loin encore ce principe de couplage faible. Un premier prototype est présenté dans le chapitre 5 pour illustrer la faisabilité de l'approche.

Pour évaluer les apports du méta-modèle, nous avons développé des nouvelles méthodes de mesure du couplage dans une composition de services qui sont présentées dans le chapitre 3. De fait, les propositions actuelles, souvent héritées des travaux dans l'objet, se sont rapidement révélées insuffisantes car ne prenant pas en compte l'ensemble des spécificités du SOSE. Nous avons donc apporté une nouvelle définition du couplage découpée en trois couplages distincts mais interdépendants : les couplages sémantique, syntaxique et physique. À partir de cette séparation, nous avons pu définir de nouvelles métriques pour l'évaluation du couplage dans une composition cible. Ces métriques sont ensuite déclinées dans un cadre de comparaison d'approches de composition dans lequel nous mesurons les contributions de notre propre travail sur le méta-modèle et le comparons à l'existant.

Au fil de notre compréhension du paradigme SOSE, nous nous sommes rendus compte de sa forte similarité avec le paradigme CBSE, en termes de problématiques, de concepts fondateurs et d'approches de résolutions. Ainsi, notre dernière contribution, présentée dans le chapitre 4, se focalise sur l'établissement clair des frontières conceptuelles entre composants et services. Pour clarifier leurs similarités et leurs spécificités, nous avons défini un cadre conceptuel de comparaison capable de les manipuler simultanément, sans favoriser l'un ou l'autre des paradigmes. Ce cadre prend aussi en compte l'orienté objet afin d'apporter un point de vue plus global de l'évolution des préoccupations du génie logiciel. À travers le triptyque produits, processus et qualités, il extrait les éléments pertinents des trois paradigmes qui déterminent leur singularité. De plus, il supporte la personnalisation de l'évaluation qualitative pour garantir son adéquation aux points de vue des utilisateurs

dont la compréhension sur une qualité varie en fonction de son domaine d'application et de leur expertise. En fin de chapitre, les catégories produit, processus et qualité sont réutilisées pour mesurer les contributions de notre méta-modèle et de notre définition du couplage. Ces mesures sont mises en relation avec l'évaluation précédente du SOSE lors de sa comparaison conceptuelle avec le CBSE et l'OO puis associées à un ensemble de perspectives de travaux futurs identifiées dans la continuité directe de leurs avancées.

Nous revenons en détails sur ces perspectives dans la section suivante, puis nous les élargissons à des ouvertures de recherche différentes.

Perspectives

Les perspectives de travaux futurs sont en rapport direct avec les limitations actuelles de nos propositions. Elles sont donc organisées suivant nos trois contributions principales : le méta-modèle de service composite, la définition du couplage faible et le cadre conceptuel de comparaison.

Enfin, une dernière section fait état d'ouvertures plus globales sur le rapport de ce travail face au Cloud computing [AFG+09, ZZ09]. En particulier, nous mettons l'accent sur une nouvelle notion associée, qui nous parait prometteuse : le principe de "*Composition as a Service*" [RLM+09, BTR10].

Méta-modèle de service composite

Les perspectives possibles d'amélioration du méta-modèle sont réparties suivant les aspects conceptuels et de réalisation.

Aspects conceptuels

Dans la section 2.2.3.3, nous avons introduit le processus d'héritage entre types de schéma de collaboration. Ce processus est un élément clé de notre gestion multitenant. Il est le support qui facilite les développements de comportements spécialisés du service composite pour faire face aux exigences particulières de certains de ses clients. Cependant, ce processus d'héritage doit encore être spécifié formellement, en particulier sur la possibilité d'un héritage entre services.

Un autre processus capital à la réification d'une composition de service en un service composite, présenté dans la section 1.4.4.2, est la génération dynamique de descriptions de services. Ce processus devra être capable de construire la description de service d'un service composite à partir des descriptions de ses constituants. Ainsi, il permettra la publication des services composites qui pourront être réutilisés de façon homogène à tout autre service. Il participera aux constructions incrémentales de compositions de composites et améliorera la réutilisation.

Aspects de réalisation

Le chapitre 5 a présenté un premier prototype du méta-modèle qui illustre les différents mécanismes introduits et cherche à en démontrer la faisabilité. Cependant, son application reste du domaine du cas d'école. Le passage de la plate-forme d'implémentation SCA Tuscany à la plate-forme FraSCAti [SMF+09] devrait apporter le support de dynamicité requis pour achever notre validation technique. Une finalité possible serait d'étendre cette plate-forme et d'y introduire notre vision du service composite comme un élément de construction disponible aux architectes, au même niveau qu'un composant SCA.

Couplage faible

Le travail effectué sur le couplage faible a deux vocations principales : l'explicitation de ce concept et l'évaluation précise de sa mesure. Une perspective proche serait d'étendre son utilisation vers un troisième axe : l'assistance au développement. Cette assistance passerait par une extension d'un outil existant de développement d'applications orientées services. Cette extension inclurait des évaluations dynamiques du couplage d'une composition en cours de modélisation. Elle identifierait les points critiques impactant significativement sur le couplage global, en prenant en compte les décisions de l'architecte et l'état courant du système en terme de services disponibles. Le but serait de proposer des alternatives de modélisation de la composition, en accord avec la population de services concrets disponibles, afin d'optimiser la minimalité du couplage.

Enfin, nous souhaitons étendre l'utilisation de notre méthode de mesure du couplage aux autres paradigmes de développements. Bien que son application directe semble intuitivement possible, il est nécessaire d'évaluer précisément le biais laissé par son orientation SOSE. En particulier, une évaluation dirigée vers les objets et les composants renforcerait notre travail d'explicitation de leurs différences avec le service.

Cadre conceptuel de comparaison

Nous identifions deux principales possibilités d'extension du cadre conceptuel : son amélioration directe ou sa réutilisation dans d'autres contextes.

Amélioration du cadre

Comme présentée dans la section 4.6.1.2, l'amélioration du cadre conceptuel de comparaison repose sur un raffinement des méthodes d'évaluation des six propriétés de l'aspect qualitatif. Un travail similaire à celui effectué sur le couplage faible doit être fait pour les cinq autres axes. Ainsi, nous pourrons offrir une mesure absolue des paradigmes OO, CBSE et SOSE, en complément de leur comparaison relative actuelle (section 4.5.2). Les évaluations qualitatives faites par les utilisateurs seront alors naturellement plus précises.

Enfin, ces nouvelles méthodes devront être capables d'évaluer les outils et technologies d'implémentation eux-mêmes. L'objectif final est de faire, si possible, la jonction entre

notre approche top-down et les approches bottom-up existantes. La communauté aurait alors à sa disposition une compréhension globale, de la vision conceptuelle des différents paradigmes à la maîtrise complète de leurs réalités d'implémentations.

Réutilisation du cadre

Le cadre conceptuel de comparaison est défini indépendamment de tout paradigme de développement. Il est donc théoriquement capable d'évaluer n'importe quel autre modèle. Dans l'annexe A, nous montrons cette capacité de réutilisation via l'évaluation d'approches prises par différentes technologies de réalisation de composants ou de services. Il serait intéressant d'étendre la comparaison entre OO, CBSE et SOSE au Cloud computing [AFG+09, ZZ09] qui, comme le SOSE, repose sur la notion de service, ou encore pour étudier la relation entre les services et les agents.

Cloud computing et Composition as a Service

Lors de notre étude du SOSE, nous nous sommes aperçus du glissement d'une partie de la communauté du service vers celle du Cloud computing. De fait, nous pouvons voir le cloud comme la possible promesse de réalisation des principes SOSE qui, à l'heure actuelle, n'a pas encore rempli tous les objectifs qu'il s'était fixé. En particulier, la problématique de composition dynamique de services, notre fenêtre d'analyse, est à moitié résolue. Son accomplissement futur semble venir de l'homogénéité du monde proposée par les différents fournisseurs de cloud. Chacun de ces fournisseurs peut offrir (ou imposer) un environnement sémantiquement cohérent qui faciliterait l'automatisation des processus de publication et de composition de services via l'emploi d'ontologies comprises par tous ses utilisateurs. Cette homogénéité est à première vue une réponse acceptable au verrou souligné dans [ES08] et un pas significatif vers le consensus.

De plus, le Cloud computing [AFG+09, ZZ09] apporte des notions additionnelles de *Software as a Service* (SaaS), *Infrastructure as a Service* (IaaS), et *Platform as a Service* (PaaS) qui extériorisent la façon de développer le logiciel et de maîtriser les ressources systèmes. Dans cette continuité, Rosenberg et *all* [RLM+09] introduisent le principe de *Composition as a Service* (CaaS) qui définit la composition comme un service à part entière, invocable et réutilisable par les clients. Son but est de proposer des compositions à partir de besoins fournis par des clients non experts. Blake et *all* reprennent ce concept de CaaS et listent ses enjeux futurs dans [BTR10].

Comme présenté dans le chapitre 4, l'approche service se base sur le concept de responsabilité propriétaire poussée à son extrême, où le client est uniquement responsable de l'utilisation du service. Cette utilisation est généralement liée à la composition de services. Le client combine alors les résultats des différents services qu'il réutilise pour construire une fonctionnalité plus complexe. Le principe de CaaS veut aller encore plus loin dans la délégation de responsabilités. Dans son approche, la composition est vue comme un service à part entière, offert dans le cloud, et disponible aux clients non experts. Ces derniers n'auraient alors plus besoin d'apprendre et de maîtriser toute la complexité du

processus de composition mais se focaliseraient uniquement sur leur domaine d'expertise et l'expression de la fonctionnalité métier voulue. Ce mode de fonctionnement peut avoir des impacts bénéfiques sur le temps de développement et la qualité du résultat.

Dans le CaaS, nous pourrions alors parler de styles de composition pour qualifier les différentes possibilités de compositions existantes et leur qualité respective. L'analyse des besoins du client devra être suffisamment fine pour extraire les services métiers nécessaires et le style de composition le plus adapté.

Ainsi, l'expérience acquise dans le SOSE et les environnements plus favorables du cloud semblent être des terrains propices à la réalisation du "tout est service".

Liste des publications de nos travaux

Revues internationales avec comité de lecture

1. Anthony Hock-koon and Mourad Oussalah, "*Composite Service MetaModel and Auto Composition*", Journal of Computational Methods in Sciences and Engineering (**JCMSE**), IOS Press, Volume 10, Issues 1-2, Pages S215-S229, **ISSN 1472-7978**.

Revues nationales avec comité de lecture

1. Anthony Hock-koon et Mourad Oussalah, "*Méta Modélisation de Service Composite*", Revue TSI La composition d'objets, de composant et de services, Juin 30, 2011 (**RSTSI**).

Conférences internationales avec comité de lecture

1. Anthony Hock-koon and Mourad Oussalah, "*The Product-Process-Quality Framework*", the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (**SEAA'11**), August 30 to September 2, 2011, Oulu, Finland. (**to be published**)
2. Anthony Hock-koon and Mourad Oussalah, "*Defining Metrics for Loose Coupling Evaluation in Service Composition*", the 7th International Conference on Service Computing (**SCC'10**), July 5-10, 2010, Miami, Florida, USA, Pages 362-369.
3. Anthony Hock-koon and Mourad Oussalah, "*Many to Many Service Discovery : a first approach*", the 4th European Conference on Software Architecture (**ECSA'10**), August 23-26, 2010, Copenhagen, Denmark, Pages 449-456.
4. Anthony Hock-koon and Mourad Oussalah, "*Specifying Loose Coupling from Existing Service Composition Approaches*", the 4th European Conference on Software Architecture (**ECSA'10**), August 23-26, 2010, Copenhagen, Denmark, Pages 464-471.

5. Anthony Hock-koon and Mourad Oussalah, "*Toward the Definition of a Many-to-Many Service Discovery Approach*", the 19th International Conference on Software Engineering and Data Engineering (**SEDE'10**), June 16-18, 2010, San Francisco, California, USA, Pages 27-32.
6. Anthony Hock-koon and Mourad Oussalah, "*Expliciting a Composite Service by a MetaModeling Approach*", the 4th International Conference on Research Challenges in Information Science (**RCIS'10**), May 19-21, 2010, Nice, France, Pages 533-544.
7. Anthony Hock-koon and Mourad Oussalah, "*Toward the Definition of the Loose Coupling Notion in a Composite Service*", the 2nd International Conference on Computer Engineering and Applications (**ICCEA'10**), March 19-21, 2010, Bali, Indonesia, Pages 339-343.
8. Anthony Hock-koon and Mourad Oussalah, "*Composite Service MetaModel*", the 18th International Conference on Software Engineering and Data Engineering (**SEDE'09**), June 22-24, 2009, Las Vegas, Nevada, USA, Pages 126-131.

Conférences nationales avec comité de lecture

1. Anthony Hock-koon et Mourad Oussalah, "*Vers une meilleure compréhension de la différence théorique entre un composant et un service*", 29ème Edition Inforsid (**Inforsid'11**), Mai 24-26, 2011, Lille, France.
2. Anthony Hock-koon et Mourad Oussalah, "*Définition du couplage faible dans une composition de services*", 29ème Edition Inforsid (**Inforsid'11**), Mai 24-26, 2011, Lille, France.
3. Anthony Hock-koon et Mourad Oussalah, "*Vers une meilleure compréhension de la composition de services par méta modélisation d'un service composite*", 4ème Conférence francophone sur les Architectures Logicielles (**CAL'10**), Mars 9-12, 2010, Pau, France.

Ateliers nationaux sans comité de lecture

1. Anthony Hock-koon et Mourad Oussalah, "*Vers l'auto composition par méta-modélisation d'un service composite*", Rétro-Ingénierie, Maintenance et Évolution des Logiciels (**RIMEL'10**), Décembre 15, 2010, Lille, France.
2. Anthony Hock-koon et Mourad Oussalah, "*Composition Dynamique de Services dans les Environnements d'Intelligence Ambiante*", Composants Objets Services : Modèles, Architectures et Langages (**COSMAL'09**), Janvier 27, 2009, Toulouse, France.

Bibliographie

[AAA08] Pascal André, Gilles Ardourel, and Christian Attiogbé. Composing components with shared services in the kmeliamodel. In *Software Composition*, pages 125–140, 2008.

[ACKM03] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer, 2003. isbn 978-3-540-44008-6.

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds : A berkeley view of cloud computing. 2009.

[Ami10] Abdelkrim Amirat. *Contribution à l'élaboration d'architectures logicielles à hiérarchies multiples*. PhD thesis, 2010. Université de Nantes.

[AO09] Abdelkrim Amirat and Mourad Oussalah. C3 : A metamodel for architecture description language based on first-order connector types. In *International Conference on Enterprise Information Systems, ICEIS*, pages 76–81, 2009.

[ARK08] Michael Ameling, Marcus Roy, and Bettina Kemme. Replication in service oriented architectures. In *International Conference on Software and Data Technologies, ICSOFT (PL/DPS/KE)*, pages 103–110, 2008.

[Att09] Christian Attiogbé. Can component/service-based systems be proved correct ? In *International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM*, pages 3–18, 2009.

[Bar06] Franck Barbier. An enhanced composition model for conversational enterprise javabeans. In *International ACM SIGSOFT Symposium on Component Based Software Engineering, CBSE*, pages 344–351, 2006.

[BBE+] Micheal Beisiegel, Dave Booz, Mike Edwards, Eric Herness, and Stephen Kinder. Software components : Coarse-grained verus fine-grained. *IBM developerWorks*.

[BCDW04] Jeremy S. Bradbury, James R. Cordy, Jurgen Dingel, and Michel Wermelinger. A survey of self-management in dynamic software architecture

specifications. In *ACM SIGSOFT Workshop on Self-Managed Systems, WOSS*, pages 28–33, 2004.

[BCL+06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java. *Softw., Pract. Exper.*, 36(11-12) :1257–1284, 2006.

[BGL07] André Bottaro, Anne Gérodolle, and Philippe Lalanda. Pervasive service composition in the home network. In *Advanced Information Networking and Applications, AINA*, pages 596–603, 2007.

[BGP07] Luciano Baresi, Sam Guinea, and Liliana Pasquale. Self-healing bpel processes with dynamo and the jboss rule engine. In *International Workshop on Engineering of Software Services for Pervasive Environments, ESSPE*, pages 11–20, 2007.

[BGPT09] Luciano Baresi, Sam Guinea, Marco Pistore, and Michele Trainotti. Dynamo + astro : An integrated approach for bpel monitoring. In *International Conference on Web Services, IEEE ICWS*, pages 230–237, 2009.

[BHI10] Jeppe Brønsted, Klaus Marius Hansen, and Mads Ingstrup. Service composition issues in pervasive computing. *IEEE Pervasive Computing*, 9 :62–70, 2010.

[BHP07] Tomas Bures, Petr Hnetynka, and Frantisek Plasil. Runtime concepts of hierarchical software components. *International Journal of Computer Information Science*, 8(S) :454–464, 2007.

[BKM07] Phil Bianco, Rick Kotermanski, and Paulo Merson. Evaluating a service-oriented architecture. *Technical Report Software Engineering Institute Carnegie Mellon*, 2007. http ://www.sei.cmu.edu/reports/07tr015.pdf.

[BL07] Hongyu Pei Breivold and Magnus Larsson. Component-based and service-oriented software engineering : Key concepts and principles. In *Euromicro Conference on Software Engineering and Advanced Applications, SEAA*, pages 13–20, 2007.

[BP08] Sandrine Beauche and Pascal Poizat. Automated service composition with adaptive planning. In *International Conference on Service Oriented Computing, ICSOC*, pages 530–537, 2008.

[Bra04] Jeremy S. Bradbury. Organizing definitions and formalisms for dynamic software architectures. In *Technical Report 2004-477, Queens' University*, 2004.

[BTR10] M. Brian Blake, Wei Tan, and Florian Rosenberg. Composition as a service. *IEEE Internet Computing*, 14 :78–82, January 2010.

[BWDP98] Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. A comprehensive empirical validation of design measures for object-oriented systems. In *IEEE METRICS*, pages 246–257, 1998.

[C03] Peltz C. Web services orchestration and choreography. *IEEE Computer*, 36 :46–52, October 2003.

[CBB07] May Chan, Judith Bishop, and Luciano Baresi. Survey and comparison of planning techniques for web services composition. *University of Pretoria, Technical Report*, 2007. `http://polelo.cs.up.ac.za/papers/Chan-Bishop-Baresi.pdf`.

[CCG+08] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. Towards self-adaptation for dependable service-oriented systems. In *Workshop on Software Architectures for Dependable Systems, WADS*, pages 24–48, 2008.

[CCG+09] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *ESEC/SIGSOFT FSE*, pages 131–140, 2009.

[CCL06] Ivica Crnkovic, Michel R. V. Chaudron, and Stig Larsson. Component-based development process and component lifecycle. In *International Conference on Software Engineering Advances, ICSEA*, page 44, 2006.

[CCSV07] Ivica Crnkovic, Michel Chaudron, Séverine Sentilles, and Aneta Vulgarakis. A classification framework for component models. In *Proceedings of the 7th Conference on Software Engineering and Practice in Sweden*, 2007.

[CDA08] Abdelghani Chibani, Karim Djouani, and Yacine Amirat. Semantic middleware for context services composition in ubiquitous computing. In *MOBILWARE*, page 9, 2008.

[CDN08] Luca Cavallaro and Elisabetta Di Nitto. An approach to adapt service requests to actual service interfaces. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, SEAMS '08, pages 129–136, 2008.

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6) :476–493, 1994.

[CK07] Soo Ho Chang and Soo Dong Kim. A service-oriented analysis and design approach to developing adaptable services. In *International Conference on Services Computing, IEEE SCC*, pages 204–211, 2007.

[CMMC08] Yuan-Chi Chang, Pietro Mazzoleni, George A. Mihaila, and David Cohn. Solving the service composition puzzle. In *International Conference on Services Computing, IEEE SCC*, pages 387–394, 2008.

[CND90] Aloysius Cornelio, Shamkant B. Navathe, and Keith L. Doty. Extending object-oriented concepts to support engineering applications. In *International Conference on Data Engineering, ICDE*, pages 220–227, 1990.

[CNP09] Luca Cavallaro, Elisabetta Di Nitto, and Matteo Pradella. An automatic approach to enable replacement of conversational services. In *International Conference on Service Oriented Computing, ICSOC/ServiceWave*, pages 159–174, 2009.

[CRZ09] Luca Cavallaro, Gianluca Ripa, and Maurilio Zuccala. Adapting service requests to actual service interfaces through semantic annotations. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, PESOS '09, pages 83–86, 2009.

[DCD06] Mikael Desertot, Humberto Cervantes, and Didier Donsez. Frogi : Fractal components deployment over osgi. In *Software Composition*, pages 275–290, 2006.

[DGD09] Stefan Dietze, Alessio Gugliotta, and John Domingue. Exploiting metrics for similarity-based semantic web service discovery. In *International Conference on Web Services, IEEE ICWS*, pages 327–334, 2009.

[DP06] Florian Daniel and Barbara Pernici. Insights into web service orchestration and choreography. *International Journal of E-Business Research, IJEBR*, 2(1) :58–77, 2006.

[DR09] Gregorio Díaz and Ismael Rodríguez. Automatically deriving choreography-conforming systems of services. In *International Conference on Services Computing, IEEE SCC*, pages 9–16, 2009.

[DS05] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *INTERNATIONAL JOURNAL ON WEB AND GRID SERVICES*, 1(1) :1–30, 2005.

[Ecl09] Eclipse. Soa tools platform : Sca tools project. 2009. http ://www.eclipse.org/stp/sca/.

[EKM07] Abdelkarim Erradi, Naveen N. Kulkarni, and Piyush Maheshwari. Service design process for reusable services : Financial services case study. In *International Conference on Service Oriented Computing, ICSOC*, pages 606–617, 2007.

[Erl05] Thomas Erl. *Service-Oriented Architecture SOA : Concepts, Technology, and Design.* Prentice Hall, 2005. isbn 978-0131858589.

[ES08] John Erickson and Keng Siau. Web services, service-oriented computing, and service-oriented architecture : Separating hype from reality. *J. Database Manag.*, 19(3) :42–54, 2008.

[FFL10] David Frank, Liana Fong, and Linh Lam. A continuous long running batch orchestration model for workflow instance migration. In *International Conference on Services Computing, IEEE SCC*, pages 226–233, 2010.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. University of California, Irvine.

[FRG08] Walid Fdhila, Mohsen Rouached, and Claude Godart. Communications semantics for wsbpel processes. In *International Conference on Web Services, IEEE ICWS*, pages 185–194, 2008.

[FYG09] Walid Fdhila, Ustun Yildiz, and Claude Godart. A flexible approach for automatic process decentralization using dependency tables. In *International Conference on Web Services, IEEE ICWS*, pages 847–855, 2009.

[GBSC09] David Garlan, Jeffrey M. Barnes, Bradley R. Schmerl, and Orieta Celiku. Evolution styles : Foundations and tool support for software architecture evolution. In *Working IEEE/IFIP Conference on Software Architecture, WICSA/ECSA*, pages 131–140, 2009.

[GMJ08] Kristof Geebelen, Sam Michiels, and Wouter Joosen. Dynamic reconfiguration using template based web service composition. In *Proceedings of the 3rd workshop on Middleware for service oriented computing*, MW4SOC '08, pages 49–54, 2008.

[GMS09] Andrew S. Grimshaw, Mark M. Morgan, and Karolina Sarnowska. Ws-naming : location migration, replication, and failure transparency support for web services. *Concurrency and Computation : Practice and Experience*, 21(8) :1013–1028, 2009.

[GMW97] David Garlan, Robert T. Monroe, and David Wile. Acme : an architecture description interchange language. In *Conference of the Centre for Advanced Studies on Collaborative Research, CASCON*, page 7, 1997.

[GNT04] Malik Ghallab, Dana Nau, and Paola Traverso. *Automated Planning : Theory and Practice*. Morgan Kaufmann, 2004. isbn 9781558608566.

[GNY04] Xiaohui Gu, Klara Nahrstedt, and Bin Yu. Spidernet : An integrated peer-to-peer service composition framework. In *International Symposium on High-Performance Distributed Computing, HPDC*, pages 110–119, 2004.

[Gru93] Thomas Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, pages 199–220, 1993. issn 1042-8143.

[GS07] Gui Gui and Paul D. Scott. Ranking reusability of software components using coupling metrics. *Journal of Systems and Software*, 80(9) :1450–1459, 2007.

[GS08] Gui Gui and Paul Scott. New coupling and cohesion metrics for evaluation of software component reusability. *International Conference for Young Computer Scientists, ICYCS*, page 1181, 2008.

[GTOS08] Olivier Le Goaer, Dalila Tamzalit, Mourad Oussalah, and Abdelhak Seriai. Evolution shelf : Reusing evolution expertise within component-based software architectures. In *International Computer Software and Applications Conference, COMPSAC*, pages 311–318, 2008.

[GZCG10] Khaled Gaaloul, Ehtesham Zahoor, François Charoy, and Claude Godart. Dynamic authorisation policies for event-based task delegation. In *International Conference on Advanced Information Systems Engineering, CAiSE*, pages 135–149, 2010.

[HC01] Georges Heineman and William Councill. *Component based software engineering : putting the pieces together*. Addison-Wesley Professional, 2001. isbn 0201704854.

[HkO10a] Anthony Hock-koon and Mourad Oussalah. Expliciting a composite service by a metamodeling approach. In *RCIS*, 2010.

[HkO10b] Anthony A. Hock-koon and Mourad Oussalah. Many to many service discovery : A first approach. In *European Conference on Software Architecture, ECSA*, pages 449–456, 2010.

[Hof93] Christine Ruth Hofmeister. *Dynamic reconfiguration of distributed applications*. PhD thesis, College Park, MD, USA, 1993. UMI Order No. GAX94-07643.

[HSD10] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. Creating context-adaptive business processes. In *International Conference on Service Oriented Computing, ICSOC*, pages 228–242, 2010.

[Jac05] Dean Jacobs. Enterprise software as service. *Queue*, 3 :36–42, July 2005.

[Jos07] Nicolai Josuttis. *SOA in Practice : The Art of Distributed System Design*. OReilly Media, 2007. isbn 978-0596529550.

[JWY+10] Lu Jin, Jian Wu, Jianwei Yin, Yin Li, and ShuiGuang Deng. Improve service interface adaptation using sub-ontology extraction. In *International Conference on Services Computing, IEEE SCC*, pages 170–177, 2010.

[Kay03] Doug Kaye. *Loosely Coupled : The Missing Pieces of Web Services*. RDS Press, 2003. isbn 978-1881378242.

[KFS06] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. Automated semantic web service discovery with owls-mx. In *International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 915–922, 2006.

[KH06] SeongKi Kim and Sang-Yong Han. Performance comparison of dcom, corba and web service. In *PDPTA*, pages 106–112, 2006.

[KKS07] Swaroop Kalasapur, Mohan Kumar, and Behrooz A. Shirazi. Dynamic service composition in pervasive computing. *Transactions on Parallel and Distributed Systems, TPDS*, 18, 2007.

[KL96] Barbara Kithchenham and Shari Lawrence. Software quality : The elusive target. *IEEE Software*, 13 :12–21, 1996.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[KRB+07] Sujatha Kuppuraju, Kanchana Rao, Hitesh A. Bosamiya, Raghu Anantharangachar, and Subhojit Mallik. Supporting heterogeneous applications with service oriented architecture. In *International Conference on Services Computing, IEEE SCC*, pages 680–681, 2007.

[LDT07] Haihua Li, Xiaoyong Du, and Xuan Tian. A wsmo-based semantic web services discovery framework in heterogeneous ontologies environment. In *International Conference on Knowledge Science, Engineering and Management, KSEM*, pages 617–622, 2007.

[LeG09] Olivier LeGoaer. *Styles d'évolution dans les architectures logicielles*. PhD thesis, 2009. Université de Nantes.

[LFW+08] Xitong Li, Yushun Fan, Jian Wang, Li Wang, and Feng Jiang. A pattern-based approach to development of service mediators for protocol mediation. In *Working IEEE/IFIP Conference on Software Architecture, WICSA*, pages 137–146, 2008.

[LKD+03] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P King, and Richard Franck. Web service level agreement wsla language specification. 2003. www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

[LL08] Jing Luo and Ying Li. Anchor semantics enabled ranking method for service discovery and integration. In *International Conference on Web Services, IEEE ICWS*, pages 732–739, 2008.

[LL09] Lei Li, Yan Wang 0002, and Ee-Peng Lim. Trust-oriented composite service selection and discovery. In *International Conference on Service Oriented Computing, ICSOC/ServiceWave*, pages 50–67, 2009.

[LML+08] Jonathan Lee, Shang-Pin Ma, Ying-Yan Lin, Shin-Jie Lee, and Yao-Chiang Wang. Dynamic service composition : a discovery-based approach. *International Journal of Software Engineering and Knowledge Engineering, IJSEKE*, 18, 2008.

[Luc02] David Luckham. *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, 2002. isbn 0201727897.

[MG09] Gavin Mulligan and Denis Gracanin. A comparison of soap and rest implementations of a service based interaction independence middleware framework. In *Winter Simulation Conference*, pages 1423–1432, 2009.

[MHB09] Karel Masek, Petr Hnetynka, and Tomás Bures. Bridging the component-based and service-oriented worlds. In *Euromicro Conference on Software Engineering and Advanced Applications, SEAA*, pages 47–54, 2009.

[Mic11] Microsoft. Distributed component object model. 2011. http ://www.microsoft.com/com/default.mspx.

[MMMP04] Stefano Modafferi, Enrico Mussi, Andrea Maurino, and Barbara Pernici. A framework for provisioning of complex e-services. In *International Conference on Services Computing, IEEE SCC*, pages 81–90, 2004.

[Mor02] Yves Mortureux. Preliminary risk analysis. *Techniques de l'ingenieur. Securite et gestion des risques*, SE2(SE4010) :SE4010.1–SE4010.10, 2002.

[MPT08] Annapaola Marconi, Marco Pistore, and Paolo Traverso. Automated composition of web services : the astro approach. *IEEE Data Eng. Bull.*, 31(3) :23–26, 2008.

[MRD08] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Viedame - flexible and robust bpel processes through monitoring and adaptation. In *ICSE Companion*, pages 917–918, 2008.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2) :46–53, 2001.

[MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Software Eng.*, 26(1) :70–93, 2000.

[MWL+08] Qian Ma, Hao Wang, Ying Li, Guotong Xie, and Feng Liu. A semantic qos-aware discovery framework for web services. In *International Conference on Web Services, IEEE ICWS*, pages 129–136, 2008.

[MZ08] Atef Mohamed and Mohammad Zulkernine. At what level of granularity should we be componentizing for software reliability ? In *High Assurance Systems Engineering Symposium, IEEE HASE*, pages 273–282, 2008.

[MZZW09] Qian Ma, Nianjun Zhou, Yanfeng Zhu, and Hao Wang. Evaluating service identification with design metrics on business process decomposition. In *International Conference on Services Computing, IEEE SCC*, pages 160–167, 2009.

[NGM+08] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike P. Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3-4) :313–341, 2008.

[NVSM07] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, and John A. Miller. Ontology driven data mediation in web services. *Int. J. Web Service Res.*, 4(4) :104–126, 2007.

[Oa99] Mourad Oussalah and all. *Génie Objet*. Lavoisier, 1999. isbn 2-7462-0029-5.

[Oa05] Mourad Oussalah and all. *Ingénierie des composants : Concepts, techniques et Outils*. Vuibert, 2005. isbn 2711748367.

[OAS04] OASIS. Uddi version 3.0.2. 2004. http ://www.uddi.org/.

[OAS07] OASIS. Web service business process execution language ws-bpel 2.0. 2007. http ://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[OAS08] OASIS. Reference architecture for service oriented architecture 1.0. April 2008. http ://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf.

[OAS09] OASIS. Service component architecture assembly model specification version 1.1. 2009. http ://www.oasis-opencsa.org/.

[OMG08] OMG. Commom object request broker architecture 3.1. 2008. http ://www.omg.org/spec/CORBA/3.1/.

[OMG11] OMG. Business process model and notation 2.0. 2011. http ://www.bpmn.org/.

[Ope09] OpenGroup. Soa source book. 2009. http ://www.opengroup.org/projects/soa-book/.

[OSG11] OSGiAlliance. Osgi service platform release 4. 2011. http ://www.osgi.org/About/WhatIsOSGi/.

[Pap08] Mike P. Papazoglou. The challenges of service evolution. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 1–15, Berlin, Heidelberg, 2008. Springer-Verlag.

[PC09] Pankesh Patel and Sanjay Chaudhary. Context aware semantic service discovery. In *SERVICES II*, pages 1–8, 2009.

[PH07] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures : approaches, technologies and research issues. *The VLDB Journal*, 16 :389–415, July 2007.

[PLS08] Heiko Pfeffer, David Linner, and Stephan Steglich. Modeling and controlling dynamic service compositions. *International Multi-Conference on Computing in the Global Information Technology, ICCGI*, 2008. isbn 978-0-7695-3275-2.

[PRF05] Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton. Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software. In *Workshop on Inter-organizational Systems and Interoperability of Enterprise Software and Applications, MIOS/INTEROP*, pages 431–441, 2005.

[PRFT07] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. Coupling metrics for predicting maintainability in service-oriented designs. *Australian Software Engineering Conference*, 0 :329–340, 2007.

[PY10] Pascal Poizat and Yuhong Yan. Adaptive composition of conversational services through graph planning encoding. In *Leveraging Applications of Formal Methods, ISoLA (2)*, pages 35–50, 2010.

[RdBM+06] Dumitru Roman, Jos de Bruijn, Adrian Mocan, Holger Lausen, John Domingue, Christoph Bussler, and Dieter Fensel. Www : Wsmo, wsml, and wsmx in a nutshell. In *Asian Semantic Web Conference, ASWC*, pages 516–522, 2006.

[RF11] Philippe Ramadour and Myriam Fakhri. Modèle et langage de composition de services. In *Inforsid*, pages 59–75, 2011.

[RHL09] Jannis Rake, Oliver Holschke, and Olga Levina. Enhancing semantic service discovery in heterogeneous environments. In *Business Information Systems, BIS*, pages 205–216, 2009.

[Rho99] John Rhoton. *Programmer's Guide to Internet Mail, SMTP, POP, IMAP and LDAP*. 1999. isbn 1-55558-212-5.

[RKL+05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Appl. Ontol.*, 1 :77–106, January 2005.

[RLM+09] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar. Towards composition as a service - a quality of service driven approach. In *International Conference on Data Engineering, ICDE*, pages 1733–1740, 2009.

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, pages 43–54, 2004.

[SD05] Zoran Stojanovic and Ajantha Dahanayake. *Service-oriented Software System Engineering Challenges And Practices*. IGI Publishing, Hershey, PA, USA, 2005. isbn 1591404274.

[SMF+09] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and Jean-Bernard Stefani. Reconfigurable sca applications with the frascati platform. In *International Conference on Services Computing, IEEE SCC*, pages 268–275, 2009.

[Smi80] Reid Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29, 1980.

[SOK05] Adel Smeda, Mourad Oussalah, and Tahar Khammaci. Madl : Meta architecture description language. In *Software Engineering Research and Applications, SERA*, pages 152–159, 2005.

[Spr11] SpringSource. Spring framework. 2011. http ://www.springsource.org/.

[SR09] Lean Shklar and Rich Rosen. *Web Application Architecture : Principles, Protocols and Practices*. John Wiley and Sons, 2009. isbn 978-0-470-51860-1.

[Sto05] Zoran Stojanovic. *A Method for Component-based and Service-Oriented Software Systems Engineering*. PhD thesis, 2005. Delft University of Technology, isbn 90-9019100-3.

[Szy02] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Professional, 2002. isbn 0201745720.

[The08] TheSeCSETeam. Service centric system engineering. *EU Integrated Project*, 2008. http ://www.secse-project.eu/.

[TIOP01] Marinos Themistocleous, Zahir Irani, Robert M. O'Keefe, and Ray J. Paul. Erp problems and application integration issues : An empirical survey. In *Hawaii International Conference on System Sciences, HICSS*, 2001.

[TWKI08] Sayed Gholam Hassan Tabatabaei, Wan M. N. Wan-Kadir, and Suhaimi Ibrahim. A comparative evaluation of state-of-the-art approaches for web service composition. In *International Conference on Software Engineering Advances, ICSEA*, pages 488–493, 2008.

[VGS+05] Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller, and Zixin Wu. The meteor-s approache for configuring and executing dynamic web processes. *LSDIS Lab, University of Georgia Technical Rport*, 2005. http ://lsdis.cs.uga.edu/projects/meteor-s/.

[VPT06] Dionissis Vassilopoulos, Thomi Pilioura, and Aphrodite Tsalgatidou. Distributed technologies corba, enterprise javabeans, web services &#8212 ; a comparative presentation. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 280–284, 2006.

[W3C04a] W3C. Owl-s : Semantic markup for web services. 2004. http ://www.w3.org/Submission/OWL-S/.

[W3C04b] W3C. Web services choreography description language version 1.0. 2004. http ://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

[W3C06] W3C. Web services choreography working group. 2006. http ://www.w3.org/2002/ws/chor/.

[W3C11a] W3C. Http - hypertext transfer protocol 1.1. 2011. http ://www.w3.org/Protocols/.

[W3C11b] W3C. Web services description language wsdl 1.1. 2011. http ://www.w3.org/TR/wsdl.

[WB09] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD Conference*, pages 889–896, 2009.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, 1991.

[Wil99] David S. Wile. Aml : An architecture meta-language. In *Automated Software Engineering, ASE*, pages 183–190, 1999.

[Wom09] Andreas Wombacher. Alignment of choreography changes in bpel processes. In *International Conference on Services Computing, IEEE SCC*, pages 1–8, 2009.

[YCR09] Liguo Yu, Kai Chen, and Srini Ramaswamy. Multiple-parameter coupling metrics for layered component-based software. *Software Quality Journal*, 17(1) :5–24, 2009.

[YP04] Jian Yang and Mike P. Papazoglou. Service components for managing the life-cycle of service compositions. *Inf. Syst.*, 29 :97–125, April 2004.

[YS07] Yoji Yamato and Hiroshi Sunaga. Context-aware service composition and component change-over using semantic web techniques. In *International Conference on Web Services, IEEE ICWS*, pages 687–694, 2007.

[ZBD+03] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *International World Wide Web Conferences, WWW*, pages 411–421, 2003.

[Zim80] Hubert Zimmermann. Osi reference model. *IEEE Transactions on Communications*, 28 :425–432, April 1980.

[ZM11] Farhana H. Zulkernine and Patrick Martin. An adaptive and intelligent sla negotiation system for web services. *IEEE T. Services Computing*, 4(1) :31–43, 2011.

[ZMCW09] Farhana H. Zulkernine, Patrick Martin, Chris Craddock, and Kirk Wilson. A policy-based middleware for web services sla negotiation. In *International Conference on Web Services, IEEE ICWS*, pages 1043–1050, 2009.

[ZMH09] Maciej Zaremba, Jacek Migdal, and Manfred Hauswirth. Discovery of optimized web service configurations using a hybrid semantic and statistical approach. In *International Conference on Web Services, IEEE ICWS*, pages 149–156, 2009.

[ZPG10] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. Disc : A declarative framework for self-healing web services composition. In *International Conference on Web Services, IEEE ICWS*, pages 25–33, 2010.

[ZZ09] Liang-Jie Zhang and Qun Zhou. Ccoa : Cloud computing open architecture. In *International Conference on Web Services, IEEE ICWS*, pages 607–616, 2009.

Annexe A

Retour au cadre conceptuel : évaluation de Fractal, SCA et OSGi

Résumé

Dans cet annexe, nous changeons de perspectives sur les paradigmes composant et service afin d'offrir un complément de compréhension au travail proposé dans le chapitre 4. Ainsi, nous réutilisons notre cadre conceptuel pour présenter une comparaison entre certaines technologies qui réalisent les principes composant et service. Nous choisissons Fractal [BCL+06] comme modèle à composants, SCA dans ces versions Tuscany [OAS09] et FraSCAti [SMF+09] comme modèles hybrides, et OSGi [OSG11] comme modèle à services.

A.1 Introduction

Le chapitre 4 a présenté le cadre conceptuel de comparaison que nous avons construit pour identifier et organiser les similarités et les spécificités des paradigmes OO, CBSE et SOSE. Les différentes catégories ont été définies dans un objectif de généralité sans favoriser l'une ou l'autre des approches. Par la suite, les évaluations menées se sont basées sur notre compréhension des paradigmes respectifs, en s'abstrayant de tous modèles ou

technologies existantes. Dans cet annexe, nous nous focalisons sur certaines technologies de réalisation de CBSE et SOSE. L'objectif est d'illustrer une réutilisation de notre cadre conceptuel et d'offrir une perspective orientée vers la réalisation des principes composant et service.

L'annexe s'organise en fonction des technologies étudiées. Les sections A.2, A.3 et A.4 abordent respectivement les technologies Fractal [BCL+06], SCA [OAS09] et OSGi [OSG11]. Chacune d'elles fait une présentation générale de l'approche puis l'évalue en aspects quantitatifs et qualitatifs. Enfin la section A.5 conclut ce travail.

A.2 Le modèle Fractal

Fractal[1] est un modèle de composant modulaire, extensible et agnostique d'un langage de programmation particulier. Il peut être utilisé pour le design, l'implémentation, le déploiement et la reconfiguration de systèmes et d'applications, allant des systèmes d'exploitation aux plate-formes intergicielles et aux interfaces graphiques. Le modèle Fractal est de faible granularité, proche de celle d'un objet, et limite ainsi les surcoûts à son utilisation.

Le modèle de composant Fractal possède un certain nombre de caractéristiques importantes telles que :

- la récursivité : les composants peuvent être encapsulés dans des composants composites ;
- la réflexivité : les composants ont les capacités d'introspection et d'intercession ;
- le partage de composant : une instance donnée d'un composant peut être incluse (partagée) par plus d'un composant.
- etc.

La table A.1 présente les produits et processus d'une architecture à composants Fractal. Les principales différences avec la table 4.2 sur la classification du CBSE en général sont :

- le *composant partagé* : exprime qu'une instance d'un composant peut être partagée par plusieurs composites ;
- les processus d'*introspection* et d'*intercession* : autorisent les (re)configurations au runtime d'un composant.

La capacité de réflexivité (processus d'introspection et d'intercession) permet les modifications, à l'exécution, des composants et des bindings entre composants. Le modèle Fractal renforce ainsi les propriétés :

- *pouvoir expressif*, en plus de la notion de composant partagé, Fractal supporte la capacité complexe de réflexivité qui reste rare en CBSE ;
- *pouvoir évolutif*, cette réflexivité apporte des moyens de modifications à l'exécution ; et
- *couplage faible*, ces modifications possibles à l'exécution rendent l'application moins dépendante des entités en exécution.

[1] http ://fractal.ow2.org/

Tab. A.1 – Fractal : produits et processus

ASPECT QUANTITATIF			
Produit	**Élément architectural simple**	**Design-time**	Type de composant primitif, Type de connecteur
		Runtime	Composant primitif, ***Composant partagé***, Connecteur
	Élément architectural composite	**Design-time**	Type de composant composite
		Runtime	Composant composite
Processus	**Dans un niveau de description**	**Design-time**	Composition horizontale
		Runtime	Appel de fonction, ***Introspection***, ***Intercession***
	Entre niveaux de description	**Design-time**	Composition verticale
		Runtime	Appel de fontion, ***Introspection***, ***Intercession***.
	Entre niveaux abstraction		Instanciation.

La figure A.1 illustre le positionnement de Fractal par rapport à notre compréhension du CBSE.

A.3 Le modèle SCA

A.3.1 SCA Tuscany

La table A.2 présente les différents produits et processus d'une architecture à composant SCA.

Ces produits et processus sont classiques au CBSE. Cependant, la particularité de SCA est d'apporter une variabilité de gestion des technologies d'implémentation sur cinq niveaux de son architecture (implémentation des composants, des interfaces, des protocoles de communication, des propriétés non fonctionnelles et du paquetage). Cette variabilité permet l'encapsulation d'entités hétérogènes et la gestion transparente de leurs collaborations.

Ainsi, SCA renforce principalement deux propriétés :

– l'*abstraction de communication*, grâce à cette gestion transparente de multiples technologies qui sont capables de dialoguer ; et

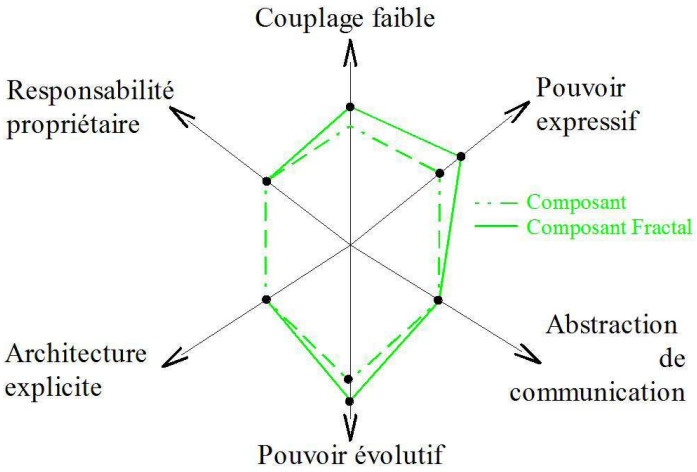

Fig. A.1 – Approche Fractal dans le CBSE

Tab. A.2 – SCA : produits et processus

ASPECT QUANTITATIF			
Produit	**Élément architectural simple**	**Design-time**	Type de composant SCA
		Runtime	Composant SCA
	Élément architectural composite	**Design-time**	Type de composite SCA
		Runtime	Composite SCA
Processus	**Dans un niveau de description**	**Design-time**	Composition horizontale
		Runtime	Appel de fonction
	Entre niveaux de description	**Design-time**	Composition verticale
		Runtime	Appel de fontion
	Entre niveaux abstraction		Instanciation

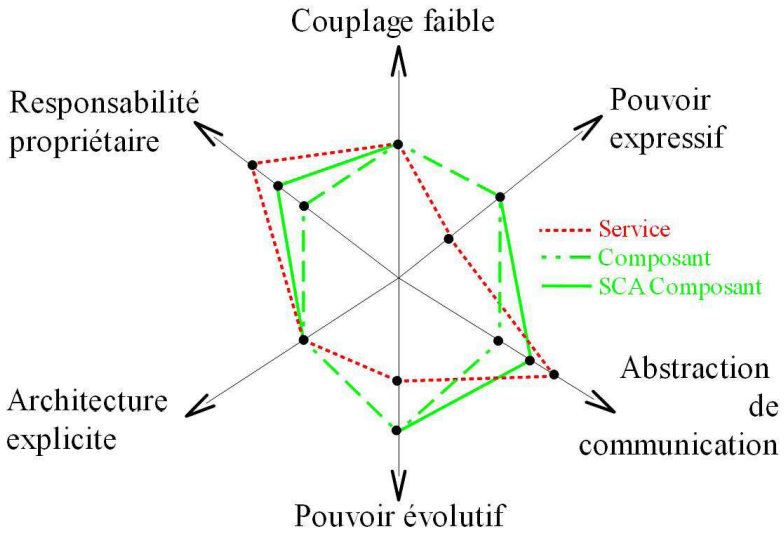

Fig. A.2 – Approche SCA : entre CBSE et SOSE

– la *responsabilité propriétaire*, grâce au support d'un des principes clés de l'approche service qu'est l'exploitation d'entités déjà déployées. En effet, lors de la construction d'une application SCA, l'architecte est capable d'intégrer des ressources déjà déployées sur le réseau via l'emploi de technologies liées aux services web (WSDL, SOAP, Rest, etc.).

La figure A.2 illustre ce rapprochement choisi par SCA entre les paradigmes CBSE et SOSE.

A.3.2 SCA FraSCAti

L'implémentation FraSCAti [SMF+09] de SCA correspond à une extension de la version classique à laquelle a été ajoutée des concepts issus de Fractal. En particulier, le principe de réflexivité, qui n'existe pas dans la version standard, permet d'améliorer la capacité de reconfigurabilité grâce à l'introspection et à l'intercession. Ainsi, le système offre les moyens de s'interroger et de se modifier en cours d'exécution. Les développeurs peuvent maintenant procéder à des reconfigurations ad hoc (non prévues à l'avance) sans arrêter l'application. De plus, ils sont capables de programmer des reconfigurations anticipées qui s'exécuteront sous les conditions prédéterminées.

La table A.3 présente les processus ajoutés par l'implémentation FraSCAti au modèle de base de SCA.

FraSCAti supporte également des technologies additionnelles sur les cinq niveaux de variabilités telles que REST, WADL, JSON-RPC, UPnP[2], OSGi, et tout un ensemble de langages de scripts. En particulier, il offre les moyens de définir des mécanismes issus de l'AOP (Aspect-oriented programming [KLM+97]).

[2]Universal Plug and Play

Tab. A.3 – SCA FraSCAti : produits et processus

ASPECT QUANTITATIF			
Produit	**Élément architectural simple**	**Design-time**	Type de composant SCA,
		Runtime	Composant SCA,
	Élément architectural composite	**Design-time**	Type de composite SCA
		Runtime	Composite SCA
Processus	**Dans un niveau de description**	**Design-time**	Composition horizontale
		Runtime	Appel de fonction, ***Introspection***, ***Intercession***
	Entre niveaux de description	**Design-time**	Composition verticale
		Runtime	Appel de fontion, ***Introspection***, ***Intercession***
	Entre niveaux abstraction		Instanciation.

Ainsi, SCA FraSCAti améliore SCA Tuscany sur de nombreux points qui sont résumés dans la figure A.3. Hormis l'*architecture* SCA et la *responsabilité propriétaire* qui restent inchangées, les autres propriétés sont perfectionnées :

- *couplage faible* : le processus de reflexivité et les nouvelles technologies supportées rendent les applications moins dépendantes des entités de réalisation ;
- *pouvoir expressif* : SCA FraSCAti supporte le processus de reflexivité et des principes issus de l'AOP ;
- *abstraction de communication* : la manipulation de technologies additionnelles rend de nouveaux supports de communications transparents ;
- *pouvoir évolutif* : la reflexivité et la gestion AOP apportent des nouveaux moyens pour les évolutions dynamiques d'une application.

A.4 Le modèle OSGi

Il n'existe pas de technologies à proprement parlé pour l'implémentation d'une architecture orientée services. Typiquement, elles sont réalisées par des combinaisons de multiples technologies en charge des différents aspects importants tels que les découvertes et compositions dynamiques de services.

Cependant, OSGi [OSG11] est un exemple d'application des concepts AOS[3] dans un

[3]Architecture Orientée Services

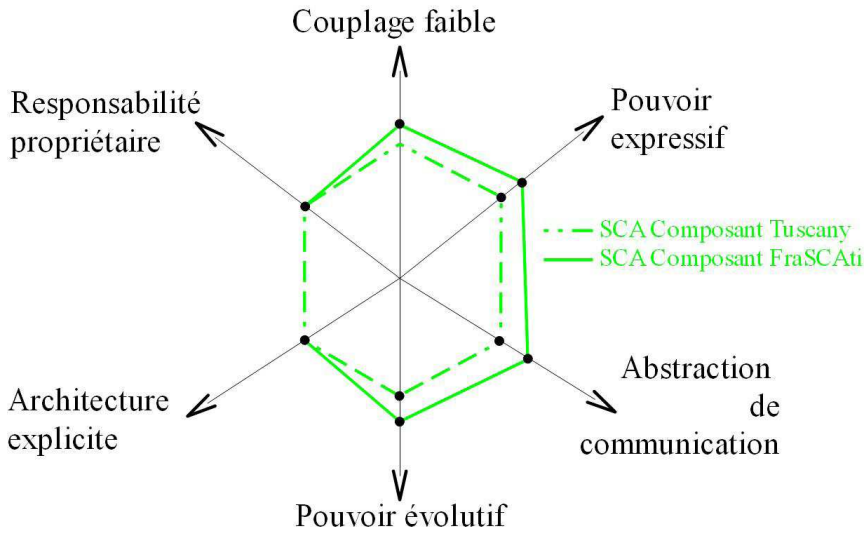

Fig. A.3 – Comparaison SCA Tuscany et SCA FraSCAti

langage de programmation. Il incorpore aux technologies Java des mécaniques classiques du paradigme service afin d'améliorer le couplage et de supporter les liaisons dynamiques. Ainsi, il manipule un registre qui répertorie les services (une ou plusieurs interfaces Java) offerts par des bundles OSGi qui sont disponibles dans le système. L'environnement OSGi supporte les apparitions et disparitions à l'exécution de ces services OSGi, ce qui permet des remplacements dynamiques (de type 1-1) via des mécanismes similaires au processus de découverte. Enfin, il manipule explicitement les versions entre services OSGi et améliore la modularité et la flexibilité des applications java.

La table A.4 présente les différents produits et processus impliqués dans le modèle OSGi. Nous nous restreignons au niveau des *bundles OSGi* et de leurs *services* que nous choisissons de représenter uniquement au runtime. De fait, nous omettons les principes de classes et d'objet Java sur lesquels reposent directement les bundles OSGI afin de simplifier la lecture et de mettre en évidence les processus similaires aux publication et découverte du SOSE que sont les processus d'*enregistrement* et de *desenregistrement de services* OSGi et de *découverte de services* OSGi.

Ainsi, OSGi supporte deux principes fondamentaux du SOSE que sont i) les découvertes et compositions des services OSGi à l'exécution et ii) les capacités de changer dynamiquement de services utilisés. Cependant, il n'aborde pas un autre principe majeur du SOSE : la gestion des hétérogénéités. De plus, OSGi ne permet pas d'appréhender d'un seul tenant l'architecture et les collaborations entre les différentes entités de l'application car il se base directement sur le paradigme objet. La figure A.4 illustre les impacts de ces absences sur les propriétés d'*abstraction de communication*, d'*architecture explicite*, de *pouvoir expressif* et de *couplage faible* par rapport à l'évaluation conceptuelle précédente du paradigme service (section 4.5.2).

Tab. A.4 – OSGi : produits et processus

ASPECT QUANTITATIF			
Produit	**Élément architectural simple**	**Design-time**	
		Runtime	Bundle OSGi, Service
	Élément architectural composite	**Design-time**	
		Runtime	
Processus	**Dans un niveau de description**	**Design-time**	
		Runtime	Appel de services, Enregistrement de services, Découverte de services, Desenregistrement de services.
	Entre niveaux de description	**Design-time**	
		Runtime	Appel de services.
	Entre niveaux abstraction		

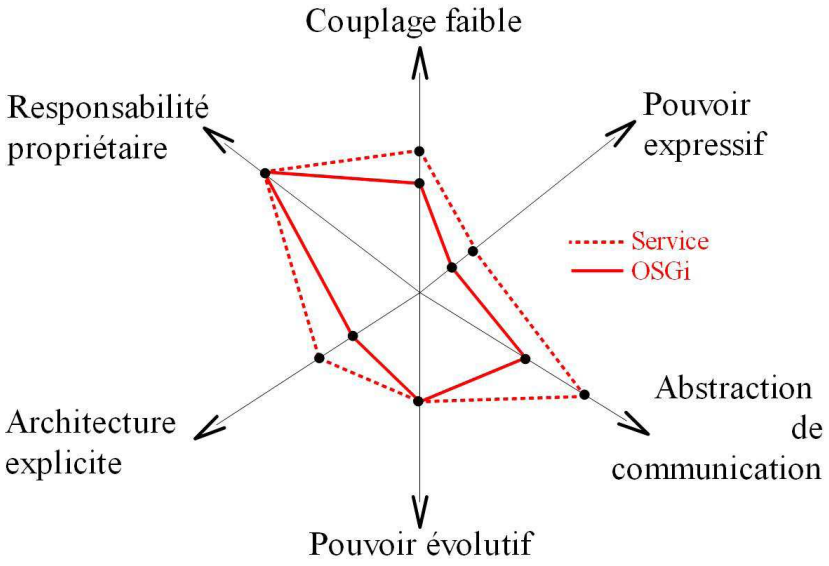

Fig. A.4 – Approche OSGi dans le SOSE

A.5 Conclusion

Dans cet annexe, nous réutilisons notre cadre conceptuel de comparaison pour montrer de quelle façon les technologies Fractal, SCA et OSGi s'inscrivent dans leur paradigme respectif. Nous soulignons les préoccupations qui ont dirigé leur construction au travers d'une évaluation qualitative suivant nos six propriétés.

Il aurait été intéressant de pouvoir combiner ces évaluations afin d'offrir une comparaison entre ces technologies. Cependant, la granularité actuelle de notre cadre ne permet d'effectuer que des comparaisons relatives, uniquement au niveau des paradigmes. Cet annexe met à nouveau en avant le besoin de raffiner nos méthodes de mesure des six propriétés afin d'offrir des évaluations absolues suffisament fines pour faire un comparatif entre technologies.

Annexe B

Découverte de services M-N : vers une première proposition

Résumé

Le processus de découverte dynamique de services est un élément essentiel du SOSE. Il est une des étapes obligatoires de la composition dynamique de services de par son rôle central dans l'identification des services concrets candidats en fonction des services abstraits. Les algorithmes de découvertes existants peuvent être catégorisés en deux approches dites 1-1 ou 1-N [KKS07]. L'approche 1-1 identifie un service abstrait à exactement un service concret pour remplir les besoins requis. L'approche 1-N identifie un service abstrait à une composition de N services concrets qui collaborent pour répondre aux exigences. Dans cet annexe, nous proposons une première piste d'étude vers une nouvelle approche de découverte de services dite M-N, où les besoins exprimés par une composition de M services abstraits peuvent être traités ensemble, et par la suite, remplis par une composition de N services concrets. Le but est d'améliorer les possibilités d'exploitations des ressources en services disponibles en proposant des solutions alternatives de réalisation non identifiées par les approches précédentes [GNY04, VGS+05, KKS07].

B.1 Introduction

Le processus de découverte de services est mentionné dans l'ensemble des chapitres de cette thèse. La compréhension de son fonctionnement, de sa place dans le paradigme SOSE et des moyens de sa réalisation ont été des passages obligés de notre état de l'art.

Comme présenté dans les chapitres 1 et 4, le processus de découverte de services est une des étapes capitales de la composition dynamique de services. Il est responsable, avec la sélection de services, du passage entre niveaux d'abstraction ; des services abstraits du design-time aux services concrets du runtime. Il supporte ainsi l'instanciation d'un type de schéma de collaboration qui assure l'exécution de la composition.

De plus, le chapitre 2 le positionne comme un processus clé des adaptations dynamiques d'un service composite. Elles le sollicitent pour déterminer des solutions de réalisation alternatives, en cas de défaillances des services concrets actuellement utilisés, et pour découvrir des services avec des fonctionnalités métiers différentes, en cas de nouveaux besoins.

Enfin, le chapitre 3 illustre l'impact du processus de découverte de services sur le couplage d'une composition. Il le localise sur le couplage syntaxique en permettant une séparation entre l'expression des besoins et les solutions de réalisation qui ne sont pas connues à l'avance. Ainsi, un service concret ne sait pas qui utilisera, ni quand et comment il sera utilisé. Notre assertion est la suivante : plus un algorithme de découverte de services est capable de fournir des solutions alternatives, moins le service composite sera dépendant de la solution courante, et plus le couplage syntaxique sera faible.

La proposition d'une première ébauche d'algorithme de découverte de type M-N va dans ce sens d'augmentation du potentiel de solutions identifiables, dans une même population de services, par rapport aux algorithmes 1-1 et 1-N existants.

L'annexe s'organise de la façon suivante. La section B.2 présente une classification des différents types d'algorithmes de découverte de services suivant cette préoccupation de correspondance entre services abstraits et services concrets. Nous y présentons les motivations de notre approche M-N, ses contributions dans le paradigme service et la façon dont ce travail s'inscrit dans la thèse. Dans la section B.3, nous définissons l'approche M-N en question et les mécaniques associées. Enfin, la section B.4 conclut et discute des possibilités futures.

B.2 Vue globale des approches de découverte de services

Comme nous l'avons présenté dans les chapitres précédents, il existe une grande diversité des approches de découvertes de services qui peuvent toutes être groupées suivant deux types de correspondances [KKS07], 1-1 et 1-N.

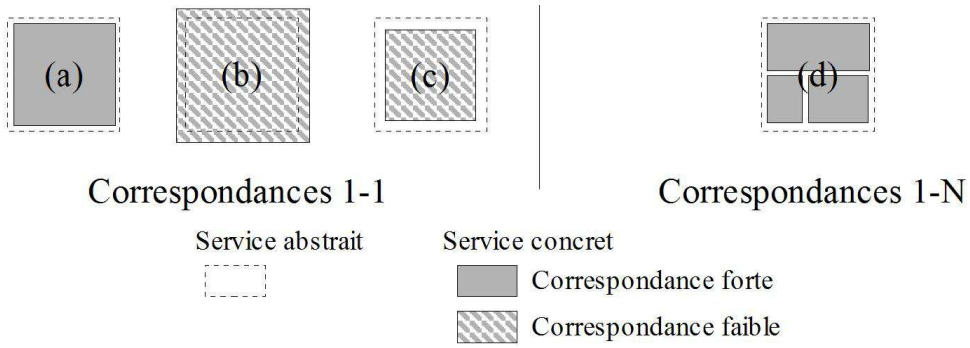

Fig. B.1 – Classification des algorithmes de découverte de services

B.2.1 Classification de l'existant

L'approche 1-1 [GNY04, VGS+05] identifie un service abstrait à exactement un service concret. Comme introduit dans la section 3.5.1.2, nous faisons une distinction entre correspondance forte et correspondance faible.

- *correspondance forte* : une exacte correspondance entre le service abstrait et le service concret, où ce dernier rempli l'ensemble des exigences métiers et de réalisation posées. (Figure B.1 (a)).
- *correspondance faible* : une correspondance non exacte, en fournissant plus (Figure B.1 (b)) ou moins (Figure B.1 (c)) de fonctions, de QoS, etc.

L'approche 1-N est une amélioration de l'approche précédente (Figure B.1 (d)). La plupart des propositions 1-N existantes se concentrent sur les contraintes d'entrées et sorties des interfaces de communications définies par le service abstrait [KKS07, BP08], c'est-à-dire les aspects données et conversationnels du service abstrait. Ainsi, elles cherchent à établir une composition de services disponibles qui puisse répondre à ces contraintes syntaxiques.

D'une manière générale, nous regroupons dans la catégorie 1-N tout approche de découvertes qui compose des solutions à partir de plusieurs services disponibles découverts dans le système. Ces solutions peuvent être encapsulées ou non dans un service composite résultat. Les approches qui utilisent un unique service découvert sont dans la catégorie 1-1. Ce cas prend aussi en compte les approches pouvant générer des entités de médiations additionnelles mais qui ne sont donc pas découvertes (section 3.5.1.2).

B.2.2 Vers une correspondance M-N

La correspondance M-N apparaît comme une amélioration naturelle des algorithmes de découvertes actuels. Elle est potentiellement capable de découvrir de nouvelles compositions de services concrets qui sont indétectables par les approches précédentes. À notre connaissance, aucun travail sur le SOSE ne supporte ce type de correspondance. Ainsi, nous proposons un ensemble de concepts pour sa réalisation et nous clarifions ses motivations.

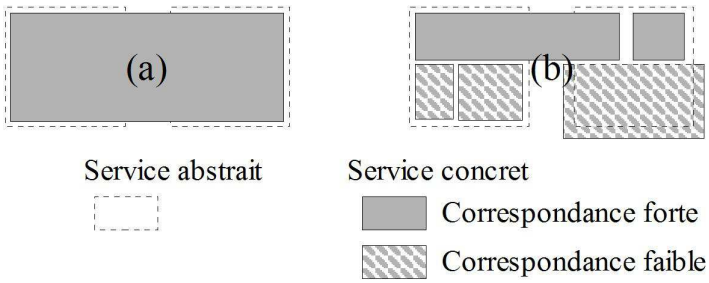

Fig. B.2 – Vers une correspondance M-N

B.2.2.1 Principes de base

Nous proposons l'introduction d'une notion de service concret "*transverse*", c'est-à-dire qui répond à des besoins métiers qui sont distribués parmi plusieurs services abstraits. La figure B.2 (a) illustre ce type de service transverse, où une composition de deux services abstraits est réalisée par un unique service concret. Ce concept est ensuite combiné avec les problématiques des correspondances fortes et faibles et des différentes compositions concrètes possibles.

Ainsi, la figure B.2 (b) illustre deux services abstraits réalisés par une composition de cinq services concrets qui inclue un service transverse. Le premier service abstrait est réalisé par trois services concrets dont le service transverse. Cependant, il y a une correspondance faible avec ce service abstrait, c'est-à-dire que ces exigences ne sont pas entièrement remplies. Le second service abstrait est aussi en correspondance faible dûe à un service concret qui apporte plus de fonctions métiers que nécessaires. Ainsi, la prise en compte de l'ensemble des possibilités de compositions par l'établissement théorique de correspondances M-N complique significativement la tâche de découverte.

B.2.2.2 Bénéfices attendus

L'objectif d'une approche M-N est la découverte de solutions concrètes additionnelles qui n'étaient pas identifiables par les approches 1-1 et 1-N précédentes. Parmi ces nouvelles solutions peut se trouver celle avec la plus haute qualité de service en rapport aux préférences utilisateurs. En augmentant le nombre d'alternatives possibles, l'étape de sélection qui suivra sera statistiquement plus sélective et la probabilité de trouver une solution adéquate est plus importante.

En plus de ces bénéfices purement quantitatifs sur le nombre des solutions, une approche M-N peut agir plus directement sur la qualité du composite. Au contraire des approches 1-1 et 1-N qui se focalisent sur les services abstraits du schéma de collaboration les uns après les autres, une approche M-N prend en compte la composition dans son ensemble. Or, la qualité d'une composition n'est pas égale à la somme des qualités individuelles de chacun des constituants mais doit être évaluée dans son ensemble. Ainsi, des métriques d'évaluation de compositions [MZZW09,PRFT07] pourraient être réutilisées pour diriger les correspondances M-N et les aider dans la sélection de la meilleure solution.

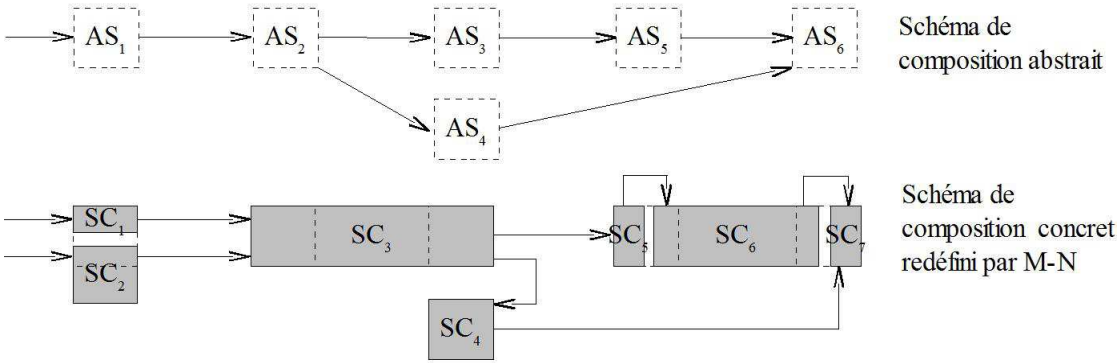

Fig. B.3 – Réorganisation de schéma de collaboration

La figure B.3 illustre un schéma de collaboration abstrait et sa réalisation concrète identifiée par une approche M-N. L'emploi de services concrets transverses modifie le découpage du schéma modélisé par l'architecte et donc les différents flots de travail et de données sont aussi modifiés. Les métriques d'évaluation de compositions peuvent être utilisées à ce moment précis pour déterminer lequel de ces découpages possibles a la meilleure qualité.

B.2.2.3 Imbrications dans les travaux de thèse

Par rapport à nos travaux précédents, cette nouvelle approche de découvertes de type M-N pourra être réutilisée à l'intérieur du gestionnaire de découverte et sélection (section 2.2.5). Elle offrirait des solutions additionnelles lors du processus d'auto-composition et participerait à la réduction du couplage syntaxique entre services abstraits et services concrets.

De plus, en travaillant directement sur le schéma de collaboration abstrait et non plus sur les services abstraits un par un, elle proposerait des modifications sur la définition même des différents services abstraits et de leur flots de contrôle. En effet, les services transverses modifient le découpage des besoins de la tâche. Ainsi, de nouveaux schémas de collaboration abstraits pourraient être déduis des solutions identifiées par l'algorithme. Ces schémas abstraits serait particulièrement utiles dans le gestionnaire de collaboration (section 2.2.3) qui les maintiendrait dans sa liste de graphes de hiérarchie d'héritage et d'instanciation comme nouvelles possibilités de réalisation de ses fonctionnalités.

B.3 Présentation de l'approche M-N

Notre piste de proposition repose sur une classification et une organisation spécifique des services concrets disponibles dans le système. Tous les services sont regroupés dans différentes *familles de services* qui sont liées entre elles par un ensemble de relations sémantiques.

B.3.1 La notion de famille de services

Une famille de services représente un ensemble de services sémantiquement proches ; c'est-à-dire ciblant les mêmes applications et utilisateurs potentiels, et partageant le même domaine d'ontologies lié à leurs activités commerçantes. Cette approche de regroupement est déjà présente dans des travaux tels que [VGS+05]. Cependant, la différence vient de la manière d'organiser ces regroupements où nous proposons des relations plus spécifiques entre ces familles pour supporter des découvertes M-N.

Ainsi, nous proposons trois relations sémantiques binaires entre familles qui seraient à étudier en profondeur : le *lien d'héritage*, le *lien d'équivalence* et le *lien d'union*.

- le *lien d'héritage* : s'inspire directement des objets où une famille A hérite d'une famille B implique de l'ontologie de domaine de A est une spécialisation de celle de B. Par exemple, une relation d'héritage entre l'ontologie "location de voitures de luxe" et la simple "location de voitures". Les services inclus dans la famille A réalisent au moins toutes les fonctionnalités attendues de la famille B. On peut parler de *lignée familiale*. Le lien d'héritage est orienté (A hérite de B).
- le *lien d'équivalence* : représente une équivalence fonctionnelle entre deux familles. Une famille A équivalente à une famille B implique que les services de A et les services de B réalisent les mêmes fonctionnalités. Cela représente une équivalence de domaine d'ontologies, par exemple une ontologie anglaise "car rental" est équivalente à la française "location de voitures". On peut parler de relation de *fraternité*.
  Le lien d'équivalence est classiquement transitif et non orienté, c'est-à-dire que si A est équivalent à B alors B est équivalent à A, et si B est équivalent à C alors A est équivalent à C.
- le *lien d'union* : représente des possibilités de compositions entre les services de deux familles différentes. Deux familles liées par un lien d'union peuvent être vues comme une nouvelle famille. On peut comprendre cette approche comme une relation *maritale* qui définit une nouvelle famille du point de vue de ce couple qu'est la composition.
  Le lien d'union est transitif, orienté, où des liens d'union entre A et B puis B et C permettent d'appréhender ces trois familles comme une seule grande famille.

Ce principe de regroupement de services suivant leur sémantique puis la définition de relations entre ces regroupements s'inspirent des travaux sur les ontologies [Gru93]. Cependant, la découverte de services ne faisant pas partie des problématiques de départ de notre thèse, nous n'avons pas pu effectuer une recherche bibliographie en profondeur sur les ontologies pour déterminer de quelle manière notre intuition peut exploiter l'existant. Nous avons donc choisi d'exprimer les concepts à haut niveau, sans rentrer dans les détails de spécification, pour permettre la présentation de notre idée de réalisation d'une correspondance M-N.

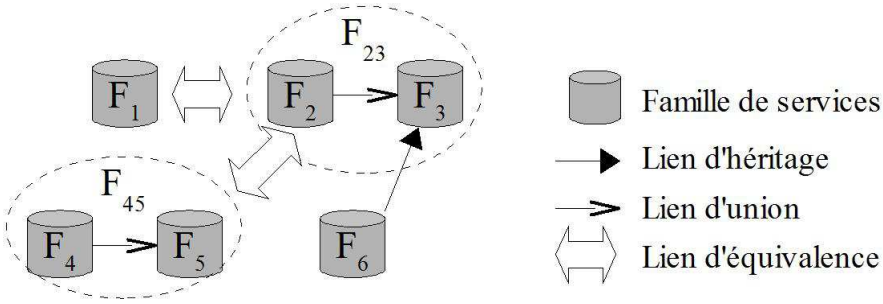

Fig. B.4 – Exemple de graphe de familles de services

B.3.2 Graphes de familles

Les trois liens sémantiques organisent les différentes familles du système. Ils permettent la construction de graphes de familles qui classifient les services disponibles.

La figure B.4 présente un exemple de graphe de familles. Ce graphe est composé de six familles. Des liens d'union associent les familles $F2$ et $F3$, et $F4$ et $F5$ qui définissent respectivement les familles $F23$ et $F45$. $F1$ est équivalent à $F23$ et $F45$ est équivalent à $F2$. $F6$ hérite de $F3$. Cette organisation particulière et la transitivité des liens assurent d'autres équivalences. Par exemple, $F2$ peut potentiellement être remplacé par l'union $F45$. $F45$ et $F3$ forment alors une nouvelle famille équivalente à $F1$. Ces successions d'équivalences sont la base d'identification de correspondances M-N.

Un graphe de familles manipule deux niveaux de relations.

- *niveau familles* : repose sur les liens d'héritage et d'équivalence qui définissent des relations sémantiques entre ontologies de domaines.
- *niveau services* : repose sur le lien d'union qui est la capacité de composer des services disponibles venant de familles différentes. C'est ce niveau qui permet l'identification des compositions concrètes.

Ces deux niveaux de relations impliquent deux niveaux de gestion : les ajouts et retraits de service dans le système, les ajouts et retraits de familles.

- **Ajouts et retraits de services** : pour être découvert et utilisé un nouveau service doit être publié dans le graphe de familles. Sa classification repose sur sa description de service et le processus de publication peut être divisé en deux étapes. Premièrement, le système doit identifier l'ontologie de domaine associée au service et l'enregistrer dans la famille correspondante. Deuxièmement, si ce nouveau service peut être composé avec des services de familles différentes alors des liens d'union entre sa famille et les autres familles doivent être définis.
  Un service qui devient indisponible ou qui est mis à l'arrêt par son fournisseur doit être retiré de la famille associée, ainsi que l'ensemble des liens d'union dépendants de sa présence.
- **Ajouts et retraits de familles** : l'ajout d'une nouvelle famille correspond à la définition de liens d'héritage ou d'équivalence avec les autres familles déjà présentes dans le graphe. Ces liens se basent sur une analyse de l'ontologie de domaine portée

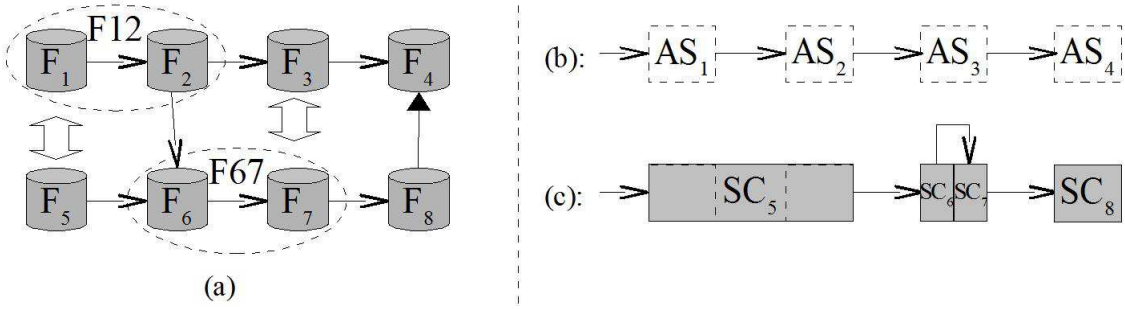

Fig. B.5 – Graphe de familles de services et réorganisation de schéma

par la nouvelle famille.
Le retrait de famille n'est pas forcément un processus pertinent. De fait, une famille exprime une certaine sémantique et un ensemble de liens qu'il peut être intéressant de conserver lors des multiples recherches d'équivalences par transitivité.

B.3.3 Identification de compositions concrètes

Les notions de familles et de liens sont utilisées pour organiser le processus de recherche de correspondances entre services abstraits et services concrets. Pour illustrer ce processus, nous utilisons un exemple simple modélisé par le graphe de famille (a) de la figure B.5 et le schéma de collaboration abstrait ciblé (b).

Tout d'abord, l'algorithme de découvertes essaie de faire une correspondance directe (1-1) entre les ontologies des services abstraits et celles des familles. Nous posons que ces correspondances sont existantes entre le service abstrait $SA1$ et la famille $F1$, puis $SA2$ et $F2$, et ainsi de suite. Après l'identification des familles appropriées, le système essaie de définir une composition entre les services concrets qu'elles classifient. Cette composition suit les liens d'unions et finalement, le système sélectionne celle qui correspond le plus aux préférences et en suivant des métriques de mesures de qualité [MZZW09, PRFT07].

Cependant, nous pouvons imaginer qu'aucune des solutions de cette correspondance directe ne remplisse les besoins et donc, des solutions alternatives doivent être identifiées. Le système se concentre maintenant sur les liens d'héritage et d'équivalence pour trouver ces solutions.

Nous posons que les familles $F1$ et $F2$ ne possèdent pas les services concrets requis. Le lien d'union qui les relie et qui est parallèle à la composition $SA1$ et $SA2$ définit la famille $F12$. Cette dernière est équivalente à la famille $F5$. Ainsi, le système cherche dans la famille $F5$ un service concret ($SC5$) pour remplir les besoins de la composition des services abstraits $SA1$ et $SA2$. Nous obtenons une correspondance 2-1.

De la même façon, ce problème peut se répéter sur la famille $F3$. En suivant le lien d'équivalence, des solutions alternatives peuvent être identifiées dans la famille résultant de l'union de $F6$ et $F7$ ($F67$). Nous obtenons une correspondance 1-2 avec la composition des services concrets $SC6$ et $SC7$ pour la réalisation du service abstrait $SA3$.

Enfin, une dernière alternative peut être proposée suivant le lien d'héritage entre $F4$ et $F8$. Les services concrets de $F8$ étant issus d'une spécialisation de fonctions, la décision de les utiliser ou non doit être laissée à l'appréciation de l'utilisateur.

Ainsi, le schéma de collaboration abstrait est redéfini en suivant le graphe de familles. La figure B.5 (c) illustre cette redéfinition.

B.3.4 Vers la définition automatisée de nouvelles familles

Notre approche de découverte M-N repose sur un graphe de famille pré-existant dans le système. La spécification des familles et de leurs relations sémantiques est dépendante d'une expertise humaine. La construction d'un graphe est difficilement automatisable. Cependant, nous voulons souligner une possibilité de méthode d'enrichissement du graphe de familles en fonction des requêtes de recherches de services demandées par un utilisateur. En effet, le schéma de collaboration abstrait est défini de façon à répondre à des buts haut niveau souhaités par son architecte. Ces buts peuvent définir une nouvelle ontologie qui peut être associée à une toute nouvelle famille. Un lien d'équivalence pourra ainsi être tissé entre cette famille et les familles existantes qui ont été sélectionnées lors de l'exécution de l'algorithme M-N pour implémenter le schéma abstrait. Enfin, la description du service composite définie par l'architecte est enregistrée dans cette nouvelle famille. Le principe est d'exploiter l'expertise des architectes qui ont défini les schémas de collaboration abstrait dans un but précis. Cependant, le système doit être suffisamment confiant dans leurs niveaux d'expertise pour exposer ces solutions comme des alternatives viables.

Dans notre exemple précédent (figure B.5), le schéma de collaboration abstrait définit une nouvelle famille. En fonction des correspondances respectives ($SA1$ et $F1$, $SA2$ et $F2$, etc.), cette nouvelle famille est donc équivalente aux familles $F1$, $F2$, $F3$, $F4$ en relation par des liens d'union.

B.4 Conclusion

Dans cet annexe, nous présentons notre vision d'une approche de découverte de services qui soit capable d'effectuer des correspondances de type M-N. L'objectif premier est d'augmenter le potentiel des alternatives de solutions concrètes qui puissent remplir les exigences des architectes. En augmentant ce potentiel nous cherchons particulièrement à réduire le couplage syntaxique.

Ainsi, nous introduisons les notions de familles de services et de liens sémantiques entre ces familles. Nous les organisons dans un graphe de familles qui assure la répartition des services concrets disponibles. Enfin, nous présentons comment exploiter ce graphe pour déterminer des correspondances de type M-N.

Les travaux futurs qui continueraient dans cette proposition sont encore très nombreux et principalement de trois natures :

– une étude bibliographique approfondie du domaine des ontologies pour établir de quelle manière nous pouvons spécifier et manipuler nos concepts de familles et de

liens sémantiques ;
- le développement d'un premier prototype pour prouver la faisabilité de l'approche ;
- la validation de l'approche par une évaluation précise de son coût. De fait, il est nécessaire de démontrer la pertinence même d'une approche M-N, c'est-à-dire si le surcoût de son exécution est compensé par des bénéfices qualitatifs suffisants de la composition résultat.

Ce travail sur une proposition de méthode de découvertes M-N a fait l'objet d'une publication en papier court dans la conférence ECSA 2010 [HkO10b].

Contribution to the understanding and modeling of the composition and loose coupling of services in Service-Oriented Architectures

Anthony Hock-koon

Abstract

Service-Oriented Software Engineering (SOSE) provides new ways to develop applications supporting highly volatile and heterogeneous environments. It is greatly inspired by earlier concepts, which have come from Object Orientation (OO) or from Component-Based Software Engineering (CBSE). SOSE follows a classical approach of software architecture with clearly-defined entities, here, services, and the relationships between said entities and brings a new dimension of dynamicity to these "for reuse" and "by reuse" architectural constructions. In this dissertation, we are interested in the dynamic service composition process which is the principal support for this reusability. Our contributions can be summarized in three axes. The first concerns the definition of a service composite metamodel which clarifies at the architectural level the characteristics of a composition of services which targets the reduction of coupling between these constituents. The second axis presents a new definition of loose coupling between services in collaboration and improves existing metrics. The final axis concerns the proposition of a conceptual comparison framework which focuses on the differences between the three paradigms: OO, CBSE, and SOSE.

Keywords: SOSE, CBSE, OO, Conceptual comparison framework, Dynamic service composition, Composite service, MetaModelling, Loose coupling, Metrics.

Contribution à la compréhension et à la modélisation de la composition et du couplage faible de services dans les architectures orientées services

Résumé

L'ingénierie logicielle orientée services (SOSE) apporte des moyens de développement d'applications supportant des environnements volatiles et hétérogènes. Elle s'inspire largement de concepts précédents, issus majoritairement de l'orienté objets (OO) ou de l'ingénierie logicielle basée composants (CBSE). Le SOSE suit une démarche classique d'architecture claire d'entités, les services, et de relations entre ces entités et apporte une nouvelle dimension de dynamicité dans ces constructions pour la réutilisation et par la réutilisation. Dans cette thèse, nous nous intéressons au processus de composition dynamique de services qui est le support principal de cette réutilisation. Nos contributions se résument en trois axes. Le premier concerne la définition d'un méta-modèle de service composite qui réifie au niveau architectural les caractéristiques d'une composition de services qui ciblent la réduction du couplage entre ces constituants. Le second axe apporte une nouvelle définition du couplage faible entre services en collaboration et améliore les métriques existantes. Le dernier axe concerne la proposition d'un cadre conceptuel de comparaison qui met en perspective les différences d'approches entre les trois paradigmes OO, CBSE et SOSE.

Mots-clés : SOSE, CBSE, OO, Cadre conceptuel de comparaison, Composition dynamique de services, Service composite, Méta-modélisation, Couplage faible, Métriques.

Discipline : Informatique
Spécialité : Génie logiciel

Laboratoire : Laboratoire d'Informatique de Nantes Atlantique.
umr cnrs 6241. 2, rue de la Houssinière, bp 92 208 – 44 322 Nantes, Cedex 3.